Universidad de las Ciencias Informáticas
Facultad 5

TRABAJO DE DIPLOMA PARA OPTAR POR EL TÍTULO DE INGENIERO EN CIENCIAS INFORMÁTICAS

Título:
Editor para el lenguaje Lista de Instrucciones.

Autor: Mario Alexander Meneses Yero

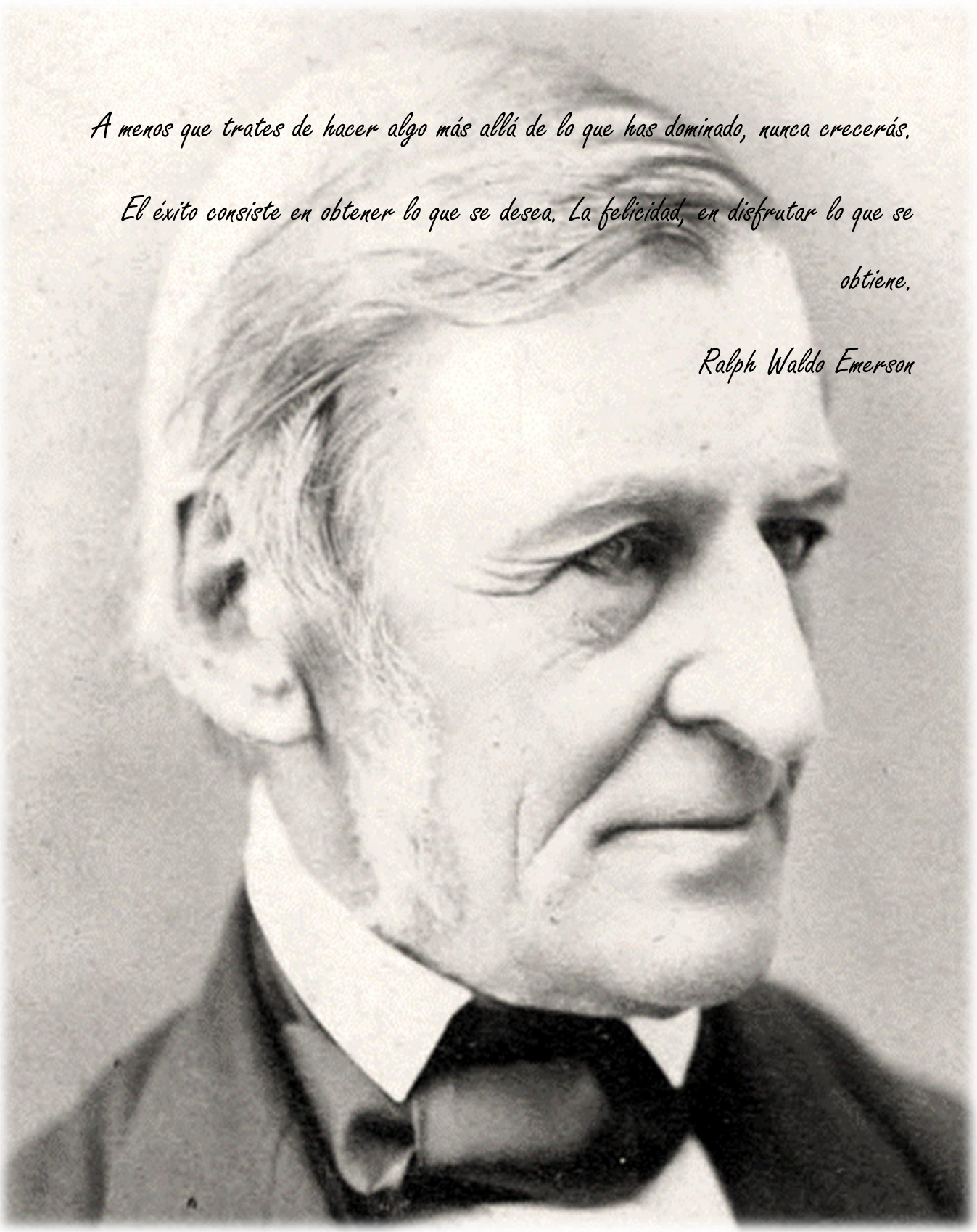
Tutor: Ing. Yunior Peralta González

Co-Tutor: Ing. Arianna Gómez Vargas

Ing. Julio Alberto Leyva Durán

Consultante: Ing. José Ernesto Carreño Bueno

PENSAMIENTO



DECLARACIÓN DE AUTORÍA

Declaro ser el autor del presente trabajo de diploma y confirmo a la Universidad de las Ciencias Informáticas los derechos patrimoniales del mismo, con carácter exclusivo. Para que así conste firmamos la presente a los ___ días del mes de _____ del año _____.

_____

Mario Alexander Meneses Yero

Autor

_____

Ing. Yunior Peralta González

Tutor

_____

Ing. Arianna Gómez Vargas

Co-tutor

_____

Ing. Julio Alberto Leyva Durán

Co-tutor

_____

Ing. José Ernesto Carreño Bueno

Consultante

DATOS DE CONTACTO

Tutor

Nombre y apellidos: Yunior Peralta González.

Institución: Universidad de las Ciencias Informáticas (UCI).

Título: Ingeniero en Ciencias Informáticas.

e-mail: yperalta@uci.cu

Co-tutor

Nombre y apellidos: Arianna Gómez Vargas.

Institución: Universidad de las Ciencias Informáticas (UCI).

Título: Ingeniero en Ciencias Informáticas.

e-mail: agomezv@uci.cu

Co-tutor

Nombre y apellidos: Julio Alberto Leyva Durán.

Institución: Universidad de las Ciencias Informáticas (UCI).

Título: Ingeniero en Ciencias Informáticas.

e-mail: jaleyva@uci.cu

Consultante

Nombre y apellidos: José Ernesto Carreño Bueno.

Institución: Universidad de las Ciencias Informáticas (UCI).

Título: Ingeniero en Ciencias Informáticas.

e-mail: jecarrenob@uci.cu

AGRADECIMIENTOS

Palabras hay muchas en el mundo, pero no creo que ninguna pueda describir por si sola lo grande que se siente este momento, tantas serían necesarias que no tendría el tiempo para citarlas todas aquí.

No les voy a mentir, mi familia es grande, y es grande porque la vida me ha dado el placer de compartirla con muchas personas que han decidido tomar parte en ella, y a todos ellos, les debo estos agradecimientos y mi gratitud. A mis abuelas, Miriam y Margarita, que me vieron crecer. A mi vieja, que la vida no me otorgo la oportunidad de demostrarle el hombre que formó después de tantos años de dolores de cabeza, momentos inolvidables y de realizar el trabajo más difícil del mundo de manera brillante, sepa, donde quiera que esté, que este momento es dedicado a ella y que siempre la llevo presente en el día a día. A mi hermanito, ese con el que siempre vivo discutiendo de cualquier cosa, que conozca que las metas son infinitas, las oportunidades ilimitadas y que siempre que se quiere, se puede. A mi viejo, que más que “mi viejo”, eres un súper padre, has hecho de padre, de madre, de amigo, de hermano, de consejero, aunque ni tú mismo no lo creas, tu apoyo y tu ejemplo, ha sido más que suficiente para quererme superar a cada segundo de mi vida. A mis tíos, Gerardo y Madelyn, por levantarme en esos momentos de derrumbe, por sus consejos, por sus regaños, por la manera en que me han enseñado a ver las cosas, más que mis tíos, son padres para mí, sepan que en mí siempre tendrán un hijo. A mis primas, Gema, Amanda y Made, que siempre con su forma de pensar aportaron su granito de arena en la persona que soy hoy. A Yailín y José Ernesto, que gracias a su apoyo y perseverancia este trabajo fue posible. A mis hermanos, sí, me refiero a esos que me han demostrado a lo largo de difíciles momentos que están y estarán ahí cuando sea que los necesite, a Felo, a Marco, a Jose, a Angel, a Leandro, a Saulito, gracias por todo, que en mí siempre tendrán un hermano. A todas mis amistades, las viejas y las nuevas, a Yanet, o “bruja”, como la llamo cariñosamente, a Yosmani, a Adrian A., a Ernesto, al Yoyi, a Silvio, a Yaikel, al Flaco, a toda la gente del grupo.

Quiero agradecerle también, a mi tutor Yunior, y mis co-tutores, Arianna y Julio A., por todo el apoyo que me han dado en esta etapa y por su guía en el desarrollo del presente trabajo. A todos los presentes, gracias.

DEDICATORIA

A mis padres, que me formaron como la persona que soy hoy, y que siempre me otorgaron su apoyo incondicional.

A mi hermano, que siempre conozca que nada es imposible y que el límite, no es hasta donde se propongan metas, sino hasta donde se desee llegar.

RESUMEN

El desarrollo de la informática ha alcanzado un elevado nivel en diversas esferas a escala mundial, entre ellas: la industria. Este avance trae consigo la utilización de determinados dispositivos para la obtención de una mayor calidad y eficiencia en los procesos productivos industriales. Nuestro país no se encuentra ajeno a esta situación, pero el coste de inversión anual, en aras de lograr la automatización de determinadas industrias, puede llegar a ser alarmante.

En el Centro de Informática Industrial (CEDIN), en la línea de Sistemas Embebidos, se funda el proyecto PLC HMI Arex; como objetivo: crear un dispositivo automático de adquisición de datos y control, como variante de automatización integral.

En la actualidad, el proyecto no cuenta con un editor para el lenguaje “Lista de Instrucciones”, que facilite la edición de rutinas para el PLC HMI Arex. La realización de dichas rutinas se lleva a cabo utilizando editores de texto del sistema, que no son la mejor opción para el proceso. Surge entonces la necesidad de crear una herramienta que permita la edición de dichas rutinas, como mecanismo de control para el dispositivo.

Al finalizar la siguiente investigación, se obtuvo como resultado una herramienta que facilita la programación del PLC HMI Arex en el lenguaje “Lista de Instrucciones”, comprendido en el estándar IEC 61131-3. A la aplicación se le realizaron varias pruebas para comprobar el correcto funcionamiento de cada una de sus funcionalidades y finalmente, será integrada en una suite denominada “Arexsoft 2.0”

Palabras claves: HMI, suite, Lista de Instrucciones.

ÍNDICE

ÍNDICE DE ILUSTRACIONES

ÍNDICE DE TABLAS

INTRODUCCIÓN

El auge de las Tecnologías de la Información y las Comunicaciones (TIC) en el presente siglo, incide sobre la mayor parte de los procesos productivos creados por el hombre, por lo que la automatización[1] de estos es necesaria. Uno de los sectores de mayor impacto del proceso de automatización es el industrial, para el cual, se han proporcionado soluciones que regulan el funcionamiento de las máquinas o elementos que intervienen en el proceso productivo, ejemplo de dichas soluciones son los Controladores Lógicos Programables (PLC, por sus siglas en inglés).

Para operar este tipo de dispositivos existen una serie de lenguajes definidos por el estándar internacional “IEC 61131”, el cual, en su tercera parte “IEC 61131-3”, define las normas de uso para cada uno de ellos.

Actualmente existen muchas herramientas para la programación de PLC, desarrolladas sobre varios tipos de tecnologías, siendo esto una de las limitaciones principales de las mismas, dado que la mayoría de estas tecnologías solo desarrollan aplicaciones propietarias para la plataforma “*Windows*”, por lo cual, es una limitación que no es capaz de suplir las necesidades del mercado actual. Hoy en día la mayoría de las empresas del mercado se ven obligadas a comprar este tipo de herramientas para la programación de sus PLC, dado que no existe una herramienta multiplataforma y libre, que pueda ser modificada según sus propias necesidades y nuevas configuraciones en sus dispositivos PLC.

Por otro lado, existen herramientas emergentes, como el Beremiz; siendo este un software multiplataforma, que permite la programación en todos los lenguajes del estándar internacional “IEC 61131-3”. Pero su interfaz gráfica es pobre, carece de facilidades para los programadores como son: completamiento de código, corrección de errores; así como fallas frecuentes (*crash fails* o fallas imprevistas), que concluyen en un cierre inesperado de la aplicación. Dejando al Beremiz, como una “herramienta primitiva” al lado de sus competidores en el mercado.

En Cuba, existen algunas herramientas de origen nacional, que permiten la manipulación de determinados dispositivos desarrollados en nuestro país. Siendo esto último su

[1]Automatización: Sistema tecnológico basado en la ingeniería y la informática, que proporciona una optimización de los procesos productivos mediante la regulación automática (autorreguladores).

principal limitante, debido que las mismas fueron desarrolladas para configurar el dispositivo para el que fueron creadas.

El Centro de Informática Industrial (CEDIN) como solución a la limitante antes mencionada, y como medida de preservación de la soberanía tecnológica, crea el proyecto PLC HMI[2] Arex, el cual tiene como objetivo desarrollar un dispositivo automático de control y adquisición de datos, utilizando una tarjeta basada en microcontroladores[3], denominada CID 300/9, desarrollada por el Instituto Central de Investigaciones Digitales (ICID). Este cuenta con una Interfaz Hombre-Máquina (HMI), que permite la visualización e interacción a los operadores con los procesos automatizados.

La rutina de control que ejecuta el PLC es programada por una persona con conocimientos de control automático, para lo cual existe la norma “IEC 61131-3” que define cuatro lenguajes de programación para este fin, entre los que se encuentra el lenguaje Lista de Instrucciones.

Actualmente en la línea Sistemas Embebidos no se cuenta con una herramienta para la programación en el lenguaje Lista de Instrucciones por lo que dicho código se edita en uno de los editores de texto multipropósitos que se halla instalado junto con en el sistema operativo, lo que dificulta el proceso de desarrollo.

Por esta razón, se plantea como **problema científico**: ¿Cómo aumentar la eficiencia del proceso de desarrollo utilizando el lenguaje Lista de Instrucciones?

Se propone como **objeto de estudio** los editores de código basado en el estándar IEC 61131.

Se define como **objetivo general** de la investigación: Desarrollar una herramienta para la edición de código en el lenguaje Lista de Instrucciones.

El **campo de acción** está constituido por los editores de código para el lenguaje Lista de Instrucciones.

Como **idea a defender** se puede enunciar: El desarrollo de un editor de programación de subrutinas de control basado en el lenguaje Lista de Instrucciones para el PLC HMI Arex.

[2] HMI: una Interfaz Hombre Máquina es la que permite que el usuario u operador del sistema de control o supervisión, interactúe con los procesos.
[3] Microcontrolador: es un circuito integrado programable, capaz de ejecutar las órdenes grabadas en su memoria.

Según lo antes planteado, se determinan las siguientes **tareas de la investigación**:

- Elaboración del marco teórico a partir de la investigación de las normas y procedimientos estandarizados para la programación de los PLC.
- Análisis del lenguaje Lista de Instrucciones de la norma internacional IEC 61131, para la programación del PLC, con el objetivo de determinar cuáles serán las instrucciones a implementar.
- Definición de las herramientas y tecnologías a utilizar para el desarrollo del editor.
- Definición de la propuesta de solución.
- Implementación del prototipo interfaz para el editor a desarrollar.
- Implementación de las funcionalidades de código seleccionadas para el lenguaje Lista de Instrucciones.
- Realización de pruebas para validar el editor desarrollado.
- Evaluación de los resultados de las pruebas realizadas.

Los **métodos de la investigación científica** utilizados son:

- **Análisis histórico-lógico**: Con el objetivo de analizar las herramientas existentes de configuración de PLC, así como su evolución cronológica, sus funcionalidades y las principales características de las mismas.
- **Análisis y síntesis**: Para realizar un estudio de la bibliografía existente y obtener de manera sintetizada el contenido necesario para darle cumplimiento al objetivo planteado.
- **Modelación**: Para crear, mediante representaciones, una visión simple de la realidad, facilitando la comprensión de la propuesta y los elementos del diseño de la solución.

Al concluir este trabajo se espera como **principal resultado**:

- Editor para la programación de subrutinas de control en el lenguaje Lista de Instrucciones para el PLC HMI Arex.

El presente documento está estructurado en tres capítulos, a continuación se describe el contenido que se abordará en cada uno de ellos:

Capítulo 1:

"Marco teórico de la investigación relacionado con la creación del sistema para la edición en el lenguaje Lista de Instrucciones"; se definen términos específicos del tema, así como conceptos y definiciones. Se realizará un análisis de diferentes herramientas, que realizan un funcionamiento similar al deseado. Serán especificadas las tecnologías y herramientas a utilizar, así como la metodología de desarrollo que se encargará de guiar todo el proceso de construcción de la solución.

Capítulo 2:

"Propuesta de solución del sistema de edición en el lenguaje Lista de Instrucciones"; se presenta la descripción de la solución propuesta a la presente situación problemática. Se describe el funcionamiento del sistema a desarrollar en relación con sus características y componentes.

Capítulo 3:

"Producción y pruebas"; se realiza una descripción de las pruebas comprendidas por la metodología aplicada, que se encargan de validar los requerimientos previstos por el cliente.

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA

Introducción

En el presente capítulo se realizará un estudio del lenguaje Lista de Instrucciones utilizado para la programación de los PLC. Además se abordarán temas relacionados con los PLC y las principales herramientas para la programación de los mismos.

1.1 Controladores Lógicos Programables.

La Comisión Electrotécnica Internacional en la norma internacional IEC 61131, publicada en el año 1992, referente a los autómatas programables y sus periféricos correspondientes, enunció la definición de PLC como: “Un sistema electrónico de funcionamiento digital, diseñado para ser utilizado en un entorno industrial, que utiliza una memoria programable para el almacenamiento interno de instrucciones orientadas al usuario para implantar unas soluciones específicas, tales como funciones de lógica, secuencia, temporizado, recuento y funciones aritméticas, con el fin de controlar mediante entradas y salidas digitales o analógicas diversos tipos de máquinas o procesos”. (2011)

Los PLC poseen dentro de su estructura, una terminal de programación encargada de la transferencia y modificación de programas, verificación de la programación e información del funcionamiento de los procesos. La que se encarga de comunicar al operario con el sistema.

1.1.1 Subrutina de control

La subrutina de control es implementada por el usuario del sistema, en uno de los lenguajes de programación de PLC, con el objetivo de ejecutar, a través del controlador, tareas de automatización y control. Es un archivo de programa separado que almacena un grupo de instrucciones y retorna el barrido del PLC a un programa principal y original. Las subrutinas se usan para organizar programas complejos o pasos repetitivos dentro de un programa.

1.2 Estándar IEC 61131.

La diversidad de especificaciones sobre los lenguajes de programación, de los fabricantes de PLC, provocaba conflictos cuando se quería establecer una comunicación entre diferentes autómatas, razón por la cual la Comisión Electrotécnica Internacional (IEC12)

crea un conjunto de instrucciones comunes como parte de un esfuerzo para alcanzar la compatibilidad de los Controladores Lógicos Programables. (Certificación, 1993)

Hasta ahora el estándar consta de las siguientes partes:

1. Información general.
2. Equipo, requerimientos y pruebas.
3. Lenguajes de programación.
4. Guías de usuario.
5. Especificación del servicio de mensajería.
6. Programación en lógica difusa.
7. Guías para aplicación e implementación de lenguajes de programación.

1.2.1 Estándar IEC 61131-3.

El estándar IEC 61131-3 consiste en la especificación de un conjunto de lenguajes de programación de PLC, haciendo énfasis en la sintaxis y semántica de cada uno. Es el primer esfuerzo real para estandarizar los lenguajes de programación usados para la automatización industrial. (Certificación, 1993)

Dentro de este estándar, en su tercera parte, se encuentran los lenguajes de programación para los PLC de forma general, los cuales poseen elementos comunes como los que se ven a continuación:

1. Tipos de Datos
2. Variables
3. Configuración, recursos y tareas
4. Unidades de organización del programa
5. Funciones

1.2.2 Lenguajes de Programación: Lista de Instrucciones

El lenguaje Lista de Instrucciones, también conocido como lenguaje booleano, utiliza la sintaxis del Álgebra de Boole para ingresar y explicar la lógica de control. Una lista de

instrucciones se compone de una secuencia de instrucciones. Cada instrucción debe comenzar en una nueva línea y debe contener un operador con modificadores opcionales, y si se precisa para la operación en particular, uno o más operadores separados por coma.

Tabla 1 Campos de instrucción.

Etiqueta	Operador	Operando	Comentario
START:	LD	%IX1	(* PULSAR BOTON *)
	ANDN	%MX5	(* NO INHIBIDOR *)
	ST	%QX2	(* CONECTAR EL VENTILADOR*)

Tabla 2 Operadores del lenguaje Lista de Instrucciones y su semántica.

Operador	Modificador	Semántica
LD	N	Poner el resultado actual igual al operando.
ST	N	Almacenar el resultado actual en el emplazamiento del operando.
S		Actualizar el operando actual a TRUE.
R		Restaurar el resultado booleano en 0.
ANDN		Negación del AND booleano.
ORN		Negación del OR booleano.
XORN		Negación del OR exclusivo booleano.
JMP	C, N	Saltar a la etiqueta.
CAL	C, N	Llamar al bloque funcional o a la función.
RET	C, N	Regresar de la función o del bloque funcional que ha sido llamado.

1.3 Estándar PLCopen

PLCopen es una organización que proporciona eficiencia independiente en la automatización industrial, basada en las necesidades de los usuarios. Miembros PLCopen se han concentrado en las especificaciones técnicas en torno a la norma IEC 61131-3, la creación de especificaciones e implementaciones con el fin de reducir el coste de la ingeniería industrial. El resultado es, por ejemplo, bibliotecas normalizadas para diferentes campos de aplicación, los niveles de conformidad de idiomas armonizada e interfaces de ingeniería para el intercambio. (Eelco van der Wal, 2009)

IEC 61131-3 se centra en el entorno de desarrollo de software. Como tal, es sólo una parte de una solución total. Las otras partes son una estructura de herramientas como:

- Herramientas de redes
- Herramientas de depuración
- Simuladores
- Las herramientas de documentación

Por lo tanto PLCopen, decide desarrollar interfaces hacia estos instrumentos de apoyo, creando el grupo de trabajo denominado TC6 para XML (*eXtended Markup Language*). Se define interfaz abierta, que es compatible con diferentes tipos de herramientas de software, y proporciona la capacidad de transferir la información que está en la pantalla para otras plataformas. La información de la pantalla no sólo contiene información textual, sino también información gráfica. Se puede incluir la posición y el tamaño de los bloques de función, y cómo están conectados. (Eelco van der Wal, 2009)

1.4 Editor de código

Un editor de código fuente es un editor de texto diseñado específicamente para editar el código fuente de programas informáticos. Puede ser una aplicación individual o estar incluido en un entorno de desarrollo integrado. (Ruiz Catalán, 2010)

Los editores de código fuente tienen características diseñadas exclusivamente para simplificar y acelerar la escritura de código fuente, como resaltado de sintaxis, autocompletar y pareo de llaves. Estos editores también proveen un modo conveniente de ejecutar un compilador, un intérprete, un depurador, o cualquier otro programa que sea relevante en el proceso de desarrollo de software. Por lo que, si bien muchos editores de texto pueden ser usados para editar código fuente sin problemas, si no mejoran, automatizan y facilitan la edición del código, no ameritan ser llamados "editores de código fuente", y son únicamente editores de texto que pueden ser usados para editar código fuente. (Ruiz Catalán, 2010)

Algunos editores de código fuente verifican la sintaxis a medida que el programador escribe, alertando inmediatamente sobre los problemas de sintaxis que puedan surgir. Otros editores de código fuente comprimen el código, convirtiendo las palabras clave en

token[4] de un solo byte, eliminando espacios en blanco innecesarios y convirtiendo los números a una forma binaria. Estos editores, descomprimen el código fuente al momento de visualizarlo, imprimiéndolo con los espacios y mayúsculas adecuadas. Existen editores que realizan ambas tareas. (Ruiz Catalán, 2010)

1.4.1 Editores de código de Lista de Instrucciones

Algunas de las herramientas especializadas en la programación de PLC son Wago I/O Pro CAA, TwidoSoft, IndraLogic, XSoft, Pc Worx Express, Beremiz. A continuación, se hace un resumen en la cual se expondrán las principales características de cada una de ellas, en aras de realizar una comparación entre las mismas.

Wago I/O Pro CAA

WAGO-I/O-PRO es una herramienta de programación y visualización. Permite el desarrollo de aplicaciones de PLC para controladores de bus de campo programables que se encuentran dentro de la WAGO-I/O-SYSTEM 750. WAGO-I/O-PRO utiliza CoDeSys Automation Alliance como una herramienta de programación. Las características incluyen: programación de los diferentes sistemas de control en una sola aplicación. Soporta los lenguajes Lista de Instrucciones (IL, por sus siglas en inglés), Diagrama de Escalera (LD, por sus siglas en inglés), Diagrama de Bloques Funcionales (FBD, por sus siglas en inglés) Texto Estructurado (ST, por sus siglas en inglés) y Diagrama Funcional (FC, por sus siglas en inglés). Indicación de estado de línea en el código del programa. Simulación offline. (WAGO, 2013)

[4]Token: Son los elementos más básicos sobre los cuales se desarrolla toda traducción de un programa.

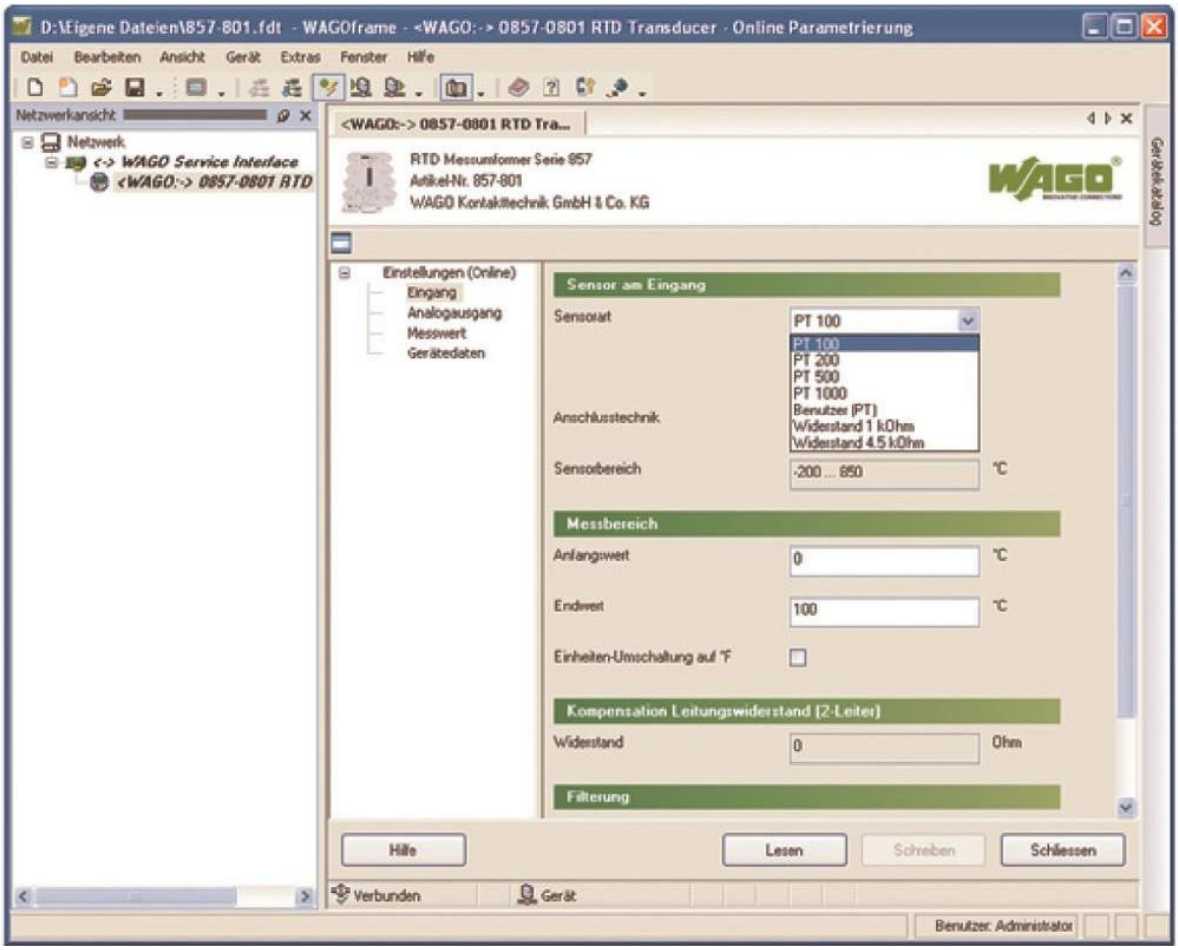


Ilustración 1 Interfaz de Wago I/O.

TwidoSoft

Es un entorno de desarrollo gráfico para crear, configurar y administrar aplicaciones para los controladores lógicos programables Twido. (Inc, 2012)

Características principales:

- La programación en Lista de Instrucciones, tanto como los lenguajes gráficos, puede ser manual (*graph set*).
- Compatible con Windows 98SE y Windows 2000 y Windows XP.
- Navegador de aplicaciones con múltiples vistas de la ventana ayudando configuración de software.
- Editores especializados (configuración, referencias cruzadas, etc.)
- Programación simbólica.
- Duplicación de los programas de aplicación.

- Compatible con PL7-07 a través de la importación ASCII.

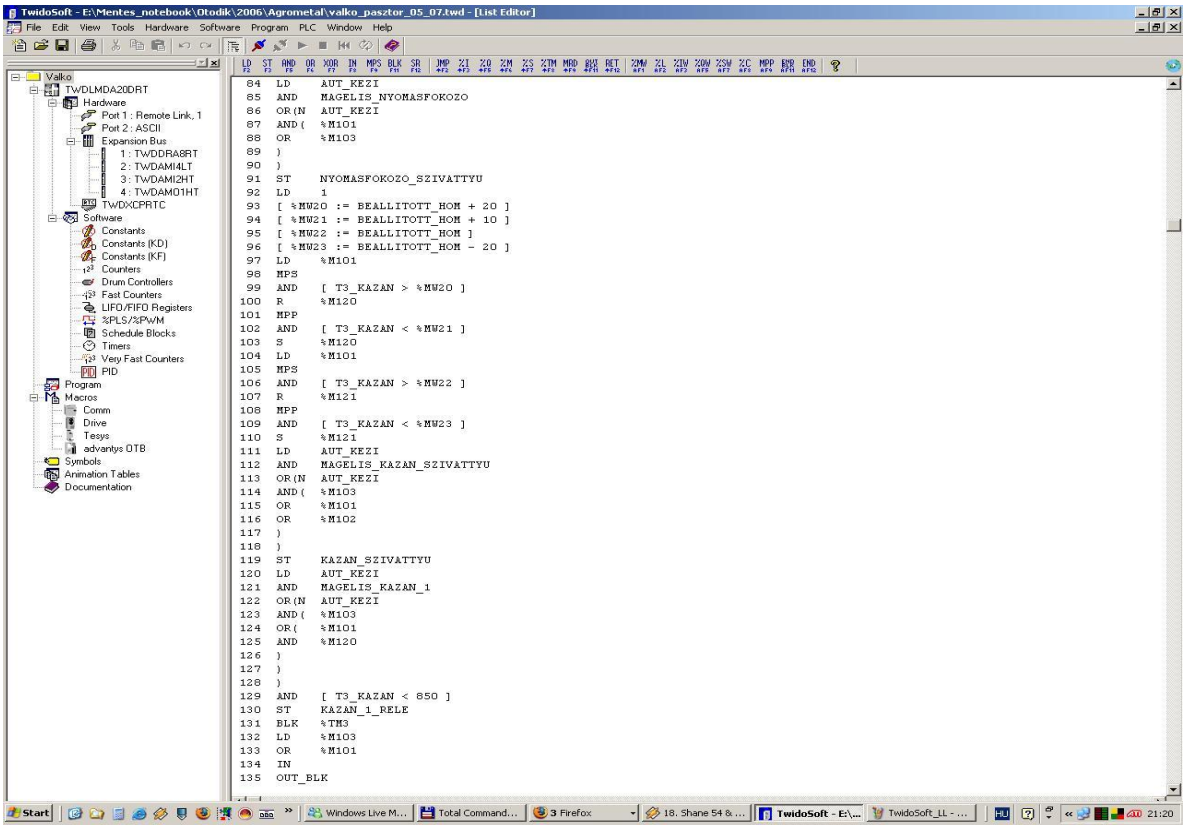


Ilustración 2 Interfaz de TwidoSoft.

IndraLogic

Es un entorno de desarrollo completo para su PLC. IndraLogic pone un enfoque simple para el potente lenguaje IEC 61131-3 a disposición del programador PLC. Rexroth IndraLogic se basa en la tecnología de CoDeSys de inteligentes soluciones de software (3S, por sus siglas en ingles). Debido a un mayor desarrollo de CoDeSys e IndraLogic, no está permitido el uso de ambos simultáneamente. Sin embargo, la compatibilidad de este con programas existentes, basados en el estándar IEC 61131-3, no se verá afectada. (Electronics, 2012)

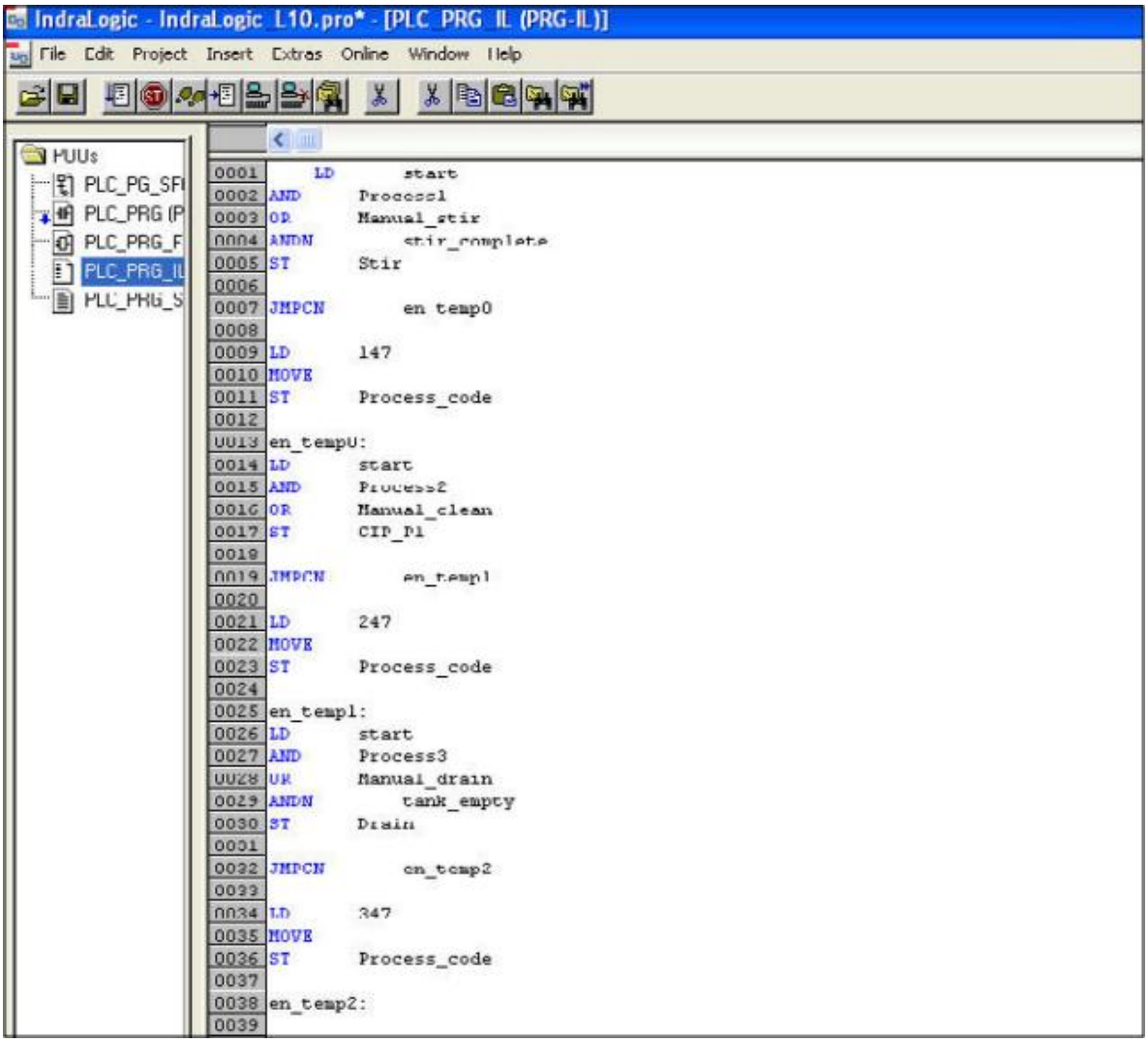


Ilustración 3 Interfaz de IndraLogic.

XSoft

Todos los controladores de *xSystem* de Eaton Automation son configurados con XSoft-CoDeSys-2. Este software está basado en la tecnología CoDeSys 3S (*Smart Software Solutions*). Características técnicas completamente desarrollados, un manejo sencillo y un amplio uso de este software en componentes de automatización de diferentes fabricantes garantizan programación exitosa con este software. (Codesys, 2013)

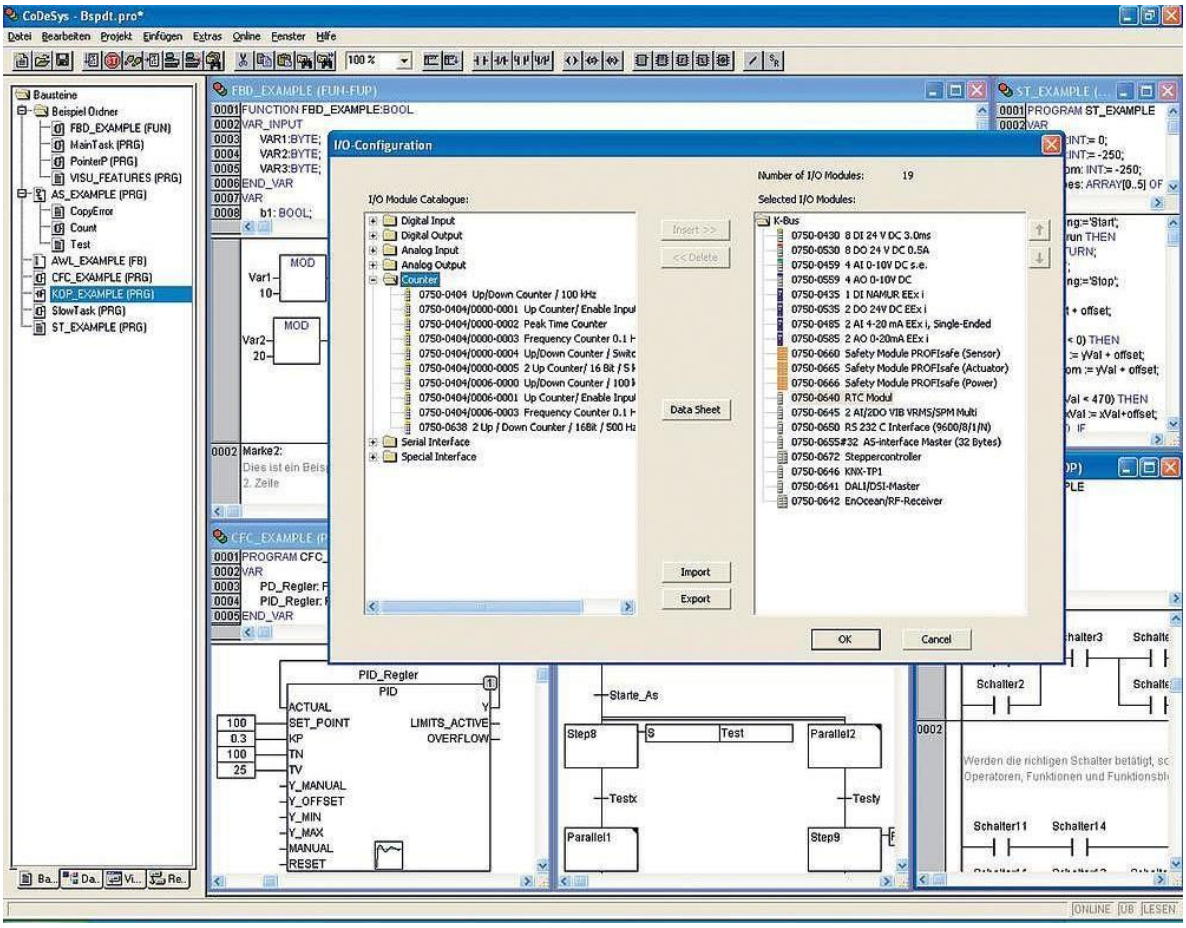


Ilustración 4 Interfaz de XSoft.

Lenguajes de programación:

- Lista de Instrucciones (IL)
- Texto Estructurado (ST)
- Diagramas de Bloques Funcionales (FBD)
- Diagrama de Función Continua (CFC)
- Diagrama de Escalera (LD)
- Diagrama de Función Secuencial (SFC)

Una serie de características simplifican la creación de aplicaciones y apoyan un objetivo: el ahorro de costes mediante la reducción de los tiempos de ingeniería. He aquí una selección de otras características: la búsqueda y reemplazo global, la generación y el uso de bibliotecas, ayuda sensible al contexto, la producción de una lista de referencias cruzadas, la comprobación de las etiquetas no utilizadas, etc.

Multitarea: La estructuración de la aplicación en tiempo de ejecución de varios programas independientes (multitarea) optimiza los recursos de su PLC y simplifica la

implementación de las tareas de tiempo crítico. Dar prioridad a los procesos de alta velocidad y proporcionar procesos más lentos con sólo la cantidad de tiempo de procesamiento según sea necesario.

Configurador de bus de campo incluido: El configurador de hardware muestra todas las E/S locales y la periferia remota (*Profibus* o *CANopen*) de una sola interfaz de usuario. Puede configurar y parametrizar las entradas y salidas directamente, y asignarles un nombre simbólico. Esto evita la ocurrencia de cualquier error de asignación entre el periférico-dispositivo y el programa del PLC. También puede probar las variables en el modo online.

PC Worx Express

Es el entorno de programación manejable, gratuito e intuitivo para todos los sistemas de mando de la clase 100. Creado por Phoenix Contact, ofrece diferentes productos de software para el sistema de autómatas compactos Easy Automation. (Contact, 2009)

La superficie de programación del software muestra una complejidad reducida en comparación con la versión íntegra PC Worx. Una programación normalizada según IEC 61131-3 simplemente posibilita la configuración de bus, la asignación E/S y el diagnóstico.

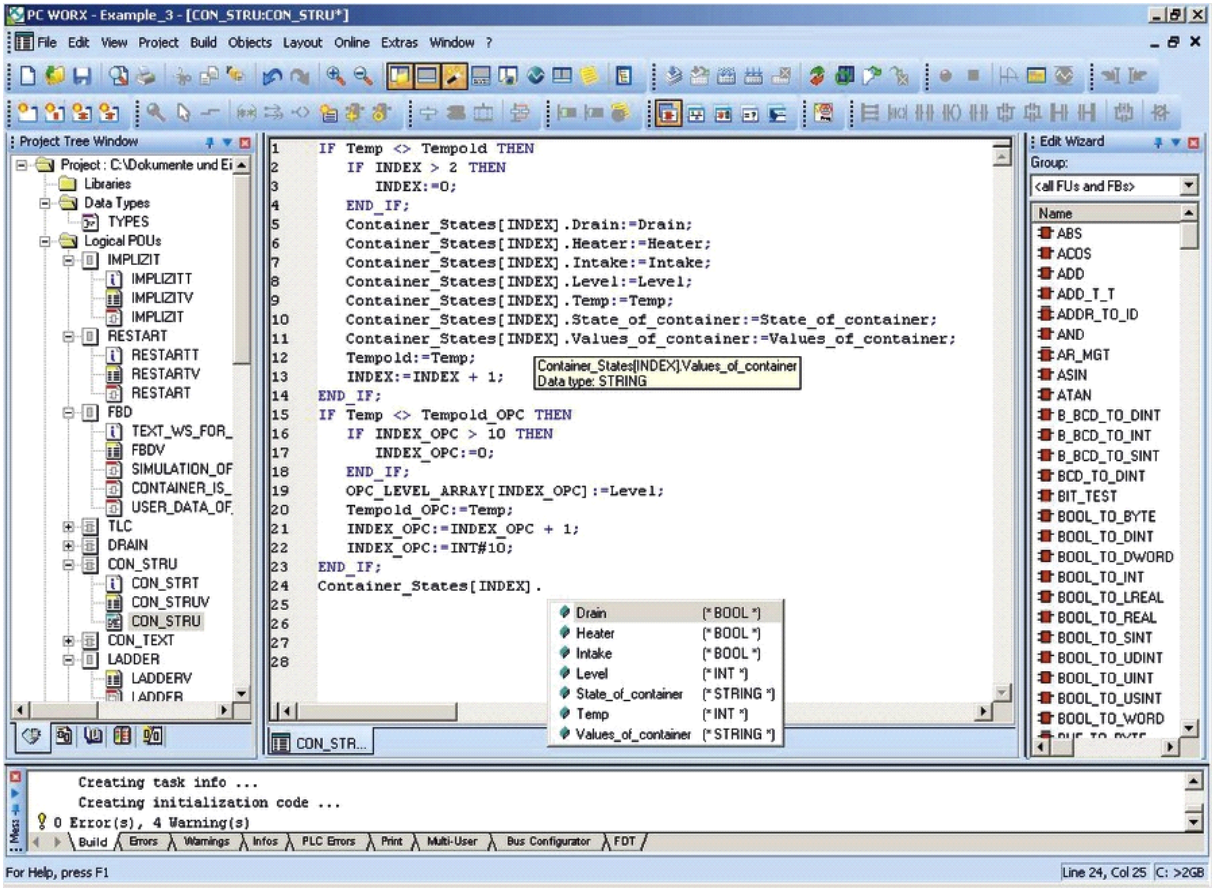


Ilustración 5 Interfaz de PC Worx Press.

PC Worx se puede utilizar para programar tareas de automatización, incluso de tipo exigente y eficiente. El software ofrece una gran variedad de funciones útiles para esto como es:

- Programación en todos los lenguajes IEC 61131-3, como la IL, FBD, LD, SFC y ST.
- Compilación cruzada de lenguajes IEC.
- Texto y gráficos editores de fácil uso.
- Ventana de referencias cruzadas.
- Asistentes de proyecto / plantilla de proyecto.
- Capacidad y protección mediante contraseña multiusuario.
- Capacidad para varios proyectos.
- Direccionamiento implícito.
- IntelliSense: herramienta de autocompletado para los códigos fuente de edición.
- Comparación de proyecto.

- Concepto del lenguaje global.

Beremiz

Está diseñado para proporcionar las herramientas necesarias para la automatización de la aplicación. Te permitirá crear interfaces y personalizarlas ofreciendo una interfaz fácil de usar. Se concentra en ser un sistema de lógica convencional y un software completo IEC 61131-3. (Inc, 2013)

Estas son algunas de las características clave de "Beremiz":

- Automatizado.
- Traduce a una gran cantidad de procesadores de los PLC.
- Uso del estándar PLCopen.
- Crea HMI personalizables.
- Respeta las normas.
- Evita el bloqueo del proveedor.

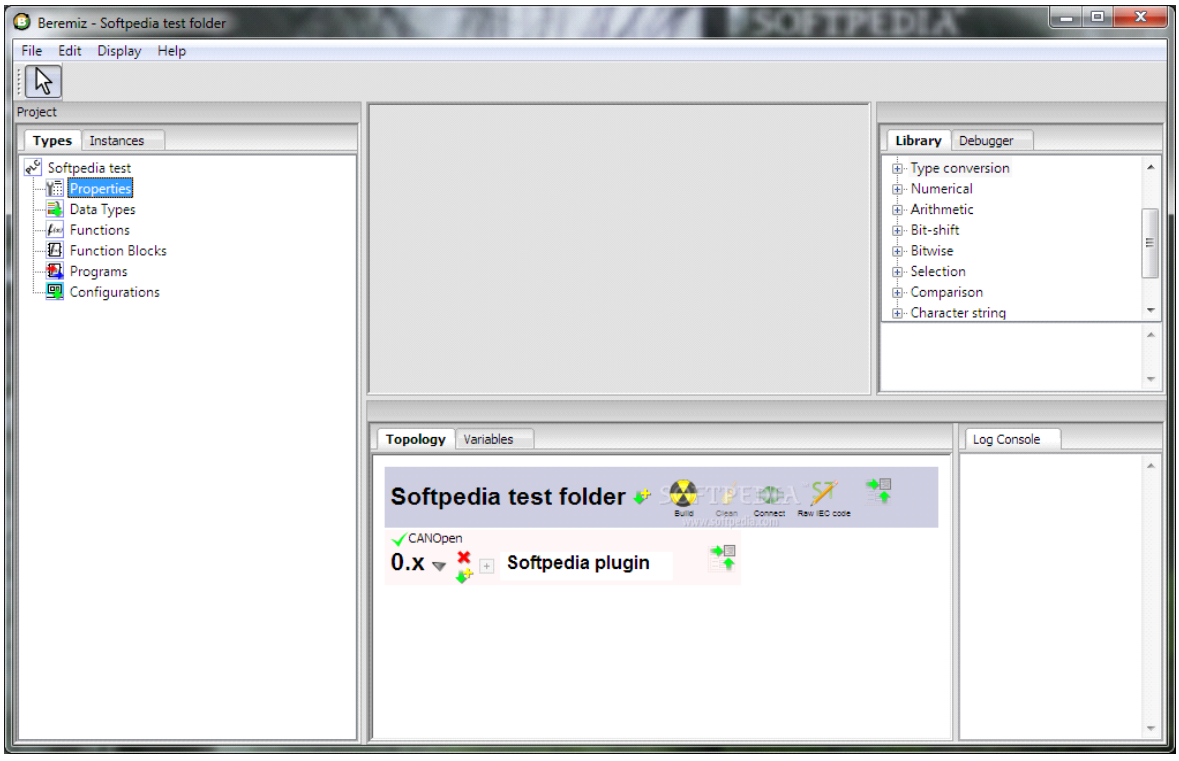


Ilustración 6 Interfaz de Beremiz.

Comparación

Tabla 3 Comparación entre los editores.

Herramienta	Software	Lenguaje de	Multiplataforma

	Propietario	**programación**	
Wago I/O Pro CAA	Si	IL, LD, FBD, ST y FC	No
TwidoSoft	Si	IL, LD, FBD, SFC y CFC	No
IndraLogic	Si	IL, ST, FBD y LD	No
XSoft	Si	IL, LD, FBD, ST, CFC y SFC	No
PC Worx Express	Si	IL, FBD, LD, SFC y ST	No
Beremiz	No	IL, ST, FBD, LD y SFC	Si

La calidad de cada una de estas herramientas es extrema, creadas con el objetivo de brindar el mejor servicio posible para la automatización de los procesos. Se puede concluir, que la única característica que comprenden los mismos antes mencionados, es que permiten la configuración de varios tipos de PLC, debido a que integran determinadas estructuras que responden a las diferentes arquitecturas de dichos PLC; aunque los cinco primeros poseen varias versiones, todas fueron desarrolladas únicamente para la plataforma "*Windows*". No siendo así para el Beremiz, siendo este multiplataforma, y, aunque todavía se trabaja en el mismo, se puede concluir que este tampoco es útil para nuestra situación, dado que posee una interfaz para la programación en el lenguaje Lista de Instrucciones un poco pobre, sin correcciones en la semántica de las instrucciones, completamiento automático, resaltado de la sintaxis, entre otros.

1.5 Selección de la metodología y herramientas

A continuación serán seleccionadas las metodologías y tecnologías compatibles con la investigación y que permitan arribar a la solución deseada. El desarrollo de software no fuera posible sin las herramientas y tecnologías, sin estas, el trabajo sería lento y tedioso para la obtención de los resultados deseados.

1.5.1 Metodología de desarrollo de software

Para asegurar el éxito durante el desarrollo de software no es suficiente contar con notaciones de modelado y herramientas, hace falta un elemento importante: la

metodología de desarrollo, la cual nos provee de una dirección a seguir para la correcta aplicación de los demás elementos. (Amaro Calderón, et al., 2010)

Generalmente el proceso de desarrollo llevaba asociado un marcado énfasis en el control del proceso mediante una rigurosa definición de roles, actividades y artefactos, incluyendo modelado y documentación detallada. Este esquema "tradicional" para abordar el desarrollo de software ha demostrado ser efectivo y necesario en proyectos de gran tamaño (respecto a tiempo y recursos), donde por lo general se exige un alto grado de ceremonia en el proceso. Sin embargo, este enfoque no resulta ser el más adecuado para muchos de los proyectos actuales donde el entorno del sistema es muy cambiante, y en donde se exige reducir drásticamente los tiempos de desarrollo pero manteniendo una alta calidad. (Rocha, 2012)

En este contexto, las metodologías ágiles emergen como una posible respuesta para llenar ese vacío metodológico. Por estar especialmente orientadas para proyectos pequeños, las Metodologías Ágiles constituyen una solución a medida para ese entorno, aportando una elevada simplificación que a pesar de ello no renuncia a las prácticas esenciales para asegurar la calidad del producto. (Rocha, 2012)

Algunas de las principales metodologías ágiles de hoy día son:

- Extreme Programming (Programación Extrema o XP)
- Scrum

XP

Es una metodología ágil centrada en potenciar las relaciones interpersonales como clave para el éxito en el desarrollo de software, promoviendo el trabajo en equipo, preocupándose por el aprendizaje de los desarrolladores, y propiciando un buen clima de trabajo. XP se basa en realimentación continua entre el cliente y el equipo de desarrollo, comunicación fluida entre todos los participantes, simplicidad en las soluciones implementadas y coraje para enfrentar los cambios. XP se define como especialmente adecuada para proyectos con requisitos imprecisos y muy cambiantes, y donde existe un alto riesgo técnico. A continuación presentaremos las características esenciales de XP organizadas en los tres apartados siguientes: historias de usuario, roles, proceso y prácticas. (Rocha, 2012)

Scrum

Es un método iterativo e incremental que enfatiza prácticas y valores de manejo de proyecto por sobre las demás disciplinas del desarrollo. Al principio del proyecto se define el *Product Backlog*, que contiene todos los requerimientos funcionales y no funcionales que deberá satisfacer el sistema a construir. Los mismos estarán especificados de acuerdo a las convenciones de la organización ya sea mediante: características, casos de uso, diagramas de flujo de datos, incidentes, tareas. El *Product Backlog* será definido durante reuniones de planeamiento con los *stakeholders*. A partir de ahí se definirán las iteraciones, conocidas como *Sprint* en el evento de Scrum, en las que se irá evolucionando la aplicación evolutivamente. Cada *Sprint* tendrá su propio *Sprint Backlog* que será un subconjunto del *Product Backlog* con los requerimientos a ser construidos en el *Sprint* correspondiente. La duración recomendada del *Sprint* es de un mes. (Blanchard, 2012)

XP vs Scrum

Ambas son metodologías ágiles, por lo que, lo que queda es hablar de la forma de trabajo de cada una:

Tabla 4 Comparación de metodologías.

Metodología	Modo de trabajo	Diseño	Pruebas
XP	Orienta el trabajo hacia los desarrolladores.	Se parte de un diseño inicial para empezar a programar, luego de que se realiza la última versión del software, se realiza una reconstrucción del diseño inicial.	Se realizan pequeñas pruebas de automatizar al final de cada versión.
Scrum	Prioriza la forma de organizar el proyecto.	Se analiza al principio de cada sprint (desarrollo por un mes) todo, lo que provoca que el avance sea lento al tener que analizar	Se realizan pruebas de baterías completas al final de cada sprint, conllevando a que estas sean más lentas a medida que

		todos los meses todo desde el inicio.	avanza el desarrollo del software.

Se selecciona la metodología XP, debido a características como son la escasa estructuración del código, la orientación a código funcional (y no documentación y demás elementos), y la posibilidad de refactorizar/reescribir el código para mejorarlo/simplificarlo.

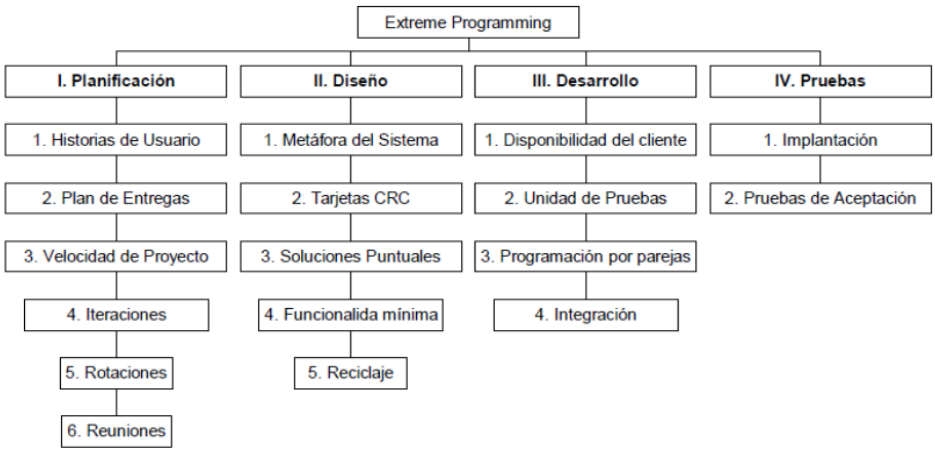


Ilustración 7 Trabajando con Extreme Programming.

1.5.2 Lenguaje de modelado

El éxito de los proyectos de desarrollo de aplicaciones o sistemas se debe a que sirven como enlace entre quien tiene la idea y el desarrollador. El UML 2.0 (*Unified Modeling Language*, Lenguaje Unificado de Modelado) es un lenguaje que cumple con esta función, ya que le ayuda a capturar la idea de un sistema para comunicarla posteriormente a quien esté involucrado en su proceso de desarrollo; esto se lleva a cabo mediante un conjunto de símbolos y diagramas. Cada diagrama tiene fines distintos dentro del proceso de desarrollo. (Schmuller, 2011)

UML 2.0 es un Lenguaje de Modelado Unificado basado en una notación gráfica la cual permite: especificar, construir, visualizar y documentar los objetos de un sistema programado. (Hernández, 2012)

El UML 2.0 modela sistemas mediante el uso de objetos que forman parte de él así como, las relaciones estáticas o dinámicas que existen entre ellos. UML puede ser utilizado por cualquier metodología de análisis y diseño orientada por objetos para expresar los diseños. (Hernández, 2012)

UML 2.0 define 13 tipos de diagramas, divididos en 3 categorías: 6 de ellos representan estructuras estáticas de la aplicación, otros 3 representan tipos generales de comportamiento y los restantes diferentes aspectos de interacción.

1.5.3 Herramienta de modelado

Hoy en día, muchas empresas se han extendido a la adquisición de herramientas *CASE* (*Computer Aided Software Engineering* o Ingeniería Asistida por Computadora), con el fin de automatizar los aspectos clave de todo el proceso de desarrollo de un sistema, desde el principio, hasta el final y así incrementar su posición en el mercado competitivo. (Sierra, 2010)

La estructura de las herramientas *CASE* se basa en la siguiente terminología: (Sierra, 2010)

- *CASE* de Alto Nivel.
- *CASE* de Bajo Nivel.
- *CASE* Cruzado de Ciclo de vida.

El *Visual Paradigm for UML* es una Herramienta *CASE* Cruzado de Ciclo de Vida. Soporta las últimas versiones del mismo, (Lenguaje de Modelado Unificado) y la Notación y Modelado de Procesos de Negocios. Desde un Grupo Administrador de Objetos. *Visual Paradigm for UML* es apoyado por un conjunto de idiomas tanto en la generación del código como en la Ingeniería Inversa por mencionar algunos ejemplos los cuales tiene la capacidad de soporte podríamos hablar de *Java*, *C++*, *CORBA IDL*, *PHP*, *XML Schema*, *Ada* y *Python*. Además, apoya la generación del código *C #*, *VB.NET*, *Object Definition Language* (ODL), *Flash Action Script*, *Delphi*, *Perl*, *C - Objective*, y *Ruby*. (Sierra, 2010)

1.5.4 Lenguaje de programación

C++ es un lenguaje artificial que puede ser usado para controlar el comportamiento de una máquina, especialmente una computadora. Estos se componen de un conjunto de reglas sintácticas y semánticas que permiten expresar instrucciones que luego serán interpretadas.” (Stroustrup, 2012)

C++ es un lenguaje de programación diseñado a mediados de los años 80 por Bjarne Stroustrup. La intención de su creación fue el extender al exitoso lenguaje de programación C con mecanismos que permitieran la manipulación de objetos. En ese

sentido, desde el punto de vista de los lenguajes orientados a objetos, el C++ es un lenguaje híbrido. Posteriormente se añadieron facilidades de programación genérica, que se sumó a los otros dos paradigmas que ya estaban admitidos (programación estructurada y la programación orientada a objetos). Por esto se suele decir que el C++ es un lenguaje de programación multiparadigma. (Stroustrup, 2012)

En la actualidad, el C++ es un lenguaje versátil, potente y general. Su éxito entre los programadores profesionales le ha llevado a ocupar un lugar dentro de los primeros puestos como herramienta de desarrollo de aplicaciones. El C++ mantiene las ventajas del C en cuanto a riqueza de operadores y expresiones, flexibilidad, concisión y eficiencia. Además, ha eliminado algunas de las dificultades y limitaciones del C original. (Stroustrup, 2012)

1.5.5 Framework de desarrollo

El marco de trabajo (*framework*) Qt vio la luz pública en el año 1995. Fue desarrollado por dos ingenieros noruegos, Haavard Nord y EirikChanble-Eng, como respuesta a la necesidad de disponer de un GUI (*Graphic User Interface* o Interfaz Gráfica de Usuario) para una aplicación C++ multiplataforma orientado a objetos. Estos ingenieros fundaron la compañía *Quasar Technologies* en 1994, nombre que fue evolucionando hasta convertirse en *Trolltech*. (Introducción al Qt y al Qt Creator, 2010)

Qt es una aplicación potente y rica, con un marco de interfaz de usuario para móviles, de escritorio y plataformas embebidas. Usando Qt, los desarrolladores pueden reutilizar cantidad significativa de código al implementar sus aplicaciones de escritorio cruzando, sistemas operativos integrados y móviles. Para algunos casos de uso, Qt permite el despliegue de plataforma cruzada con plena y sin ningún cambio en el código fuente. (Introducción al Qt y al Qt Creator, 2010)

Qt posee un potente editor de código para C++, con características como:

- El resaltado de sintaxis y completamiento automático de código.
- Comprobando de código estático mientras se escribe.
- Ayuda sensible al contexto.
- Plegado de código.
- Paréntesis partidos y completamiento de paréntesis.
- Capacidades de edición avanzadas, como la selección de bloques.

1.5.6 Entorno integrado de desarrollo

Se selecciona como entorno integrado de desarrollo (IDE, *Integrated Development Environment*) Qt Creator.

Qt Creator es un IDE creado por *Trolltech* para el desarrollo de aplicaciones con las bibliotecas Qt, requiriendo su versión 4.x. Está disponible para los sistemas operativos Linux, Max y Windows, permitiendo al desarrollador crear aplicaciones para múltiples sistemas o plataformas móviles. La naturalidad que este posee en su trabajo en conjunto junto al marco de trabajo Qt y al lenguaje de programación C++ es sin lugar a dudas algo maravilloso, tanto así, que la mayoría de las opiniones entre los programadores de la red, dictan que Qt Creator llegó para quedarse. (Introducción al Qt y al Qt Creator, 2010)

1.5.7 Bibliotecas

QScintilla 2

Además de características que se encuentran en los componentes de edición de texto estándar, QScintilla 2 incluye características especialmente útiles para la edición y depuración del código fuente. Estas incluyen soporte para el estilo de la sintaxis, indicadores de errores, completamiento de código e información sobre llamadas a funciones. El margen de selección puede contener marcadores como los utilizados en los depuradores para indicar los puntos de interrupción y la línea actual. Presenta amplias opciones de estilo como son el uso de fuente proporcional, negrita y cursiva, múltiples colores de primer plano y de fondo y varias fuentes. Está disponible bajo la licencia GNU GPL (v2 y v3), lo que constituye una limitante para su uso, debido a que obliga que su distribución contenga el código fuente.

Boost Spirit 2.5.1

Spirit es un conjunto de bibliotecas de C++ para la generación de análisis y producción implementados. Las bibliotecas Spirit permiten una gramática objetivo para ser escrito exclusivamente en C++. Especificaciones de gramática en línea pueden mezclar libremente con otro código C++ y, gracias a la potencia generativa de plantillas de C++, son de ejecución inmediata. (2012)

La biblioteca Boost::Spirit contiene una impresionante suite de clases para hacer la creación de programas de análisis orientado a objetos, tanto rápidas y limpias. Tiene la ventaja de ser muy rápida y fácil de usar sin necesidad de archivos de gramática de pre-

proceso o código de auto-generar el uso de herramientas externa. Con el fin de implementar la biblioteca, los programadores han utilizado muchas características de vanguardia de los estándares de ANSI C++, muchos de los cuales apenas están empezando a llegar a los principales compiladores. (Programmers, 2010)

Conclusiones parciales:

Se abordaron los diferentes conceptos sobre los PLC, que ayudaron a la caracterización de los mismos. El estudio del estándar IEC 61131-3, permitió definir los elementos generales para los lenguajes de programación de PLC del estándar, lo que constituyó un factor importante en el estudio del lenguaje IL.

Tras el análisis de las diferentes herramientas para la programación de PLC como son Wago I/O Pro CAA, XSoft, PC Worx Press y Beremiz; se concluyó que no es posible adaptar ninguna de las anteriores a nuestro editor, ya sea por problemas de compatibilidad con determinadas características del dispositivo que poseemos o por la plataforma en la que se utilizan las mismas. Se selecciona la metodología XP para guiar el proceso de desarrollo de software, seleccionando el IDE de programación Qt Creator, con el framework Qt y el lenguaje de programación C++. Como lenguaje de modelado UML 2.0 y herramienta de modelado Visual Paradigm for UML.

CAPÍTULO 2: PLANIFICACIÓN Y DISEÑO

Introducción

El objetivo del siguiente capítulo es describir el proceso de construcción de la solución, siguiendo la metodología de desarrollo XP. Se muestran las historias de usuarios que fueron escritas por el cliente que regirán el desarrollo de la solución, se muestran tablas y diagramas que describen el funcionamiento del sistema.

2.1 Historias de usuario

Las historias de usuario son un conjunto de fichas escritas por el cliente que indican las funciones que debe realizar el sistema, constituyendo el mecanismo base de captura de requerimientos de la programación extrema. Pueden ser creadas durante las conversaciones con los usuarios interesados (*stakeholders*) sobre nuevas funcionalidades o mejoras del proyecto. Constituyen una manera simple de describir una tarea concisa que aporta valor al usuario o al negocio. Son conformadas por los usuarios y consiste en la creación de una tarjeta que hace referencia a determinado requisito funcional o no funcional que, se desea, se incluido, modificado o reemplazado dentro del proyecto en función.

A continuación, se presenta el conjunto de historias de usuario que estarán comprendidas dentro del software.

Tabla 5 Historia de usuario № 1.

Historia de usuario	
Número: 1	**Usuario: Línea de Sistemas Embebidos**
Nombre de historia: Edición de la rutina IL.	
Prioridad del negocio: Alta	**Riesgo en el desarrollo: Alta**
Puntos estimados: 2	**Iteración asignada: 1**
Programador responsable: Mario Alexander Meneses Yero	
Descripción: Al crear el archivo IL, el usuario será capaz de programar la rutina de control en el lenguaje Lista de Instrucciones. En la ventana de edición, se resaltan en colores las palabras reservadas del lenguaje.	

Tabla 6 Historia de usuario № 2.

Historia de usuario

Número: 2	**Usuario: Línea de Sistemas Embebidos**
Nombre de historia: Realización del análisis léxico a la rutina IL en edición.	
Prioridad del negocio: Alta	**Riesgo en el desarrollo: Alto**
Puntos estimados: 3	**Iteración asignada: 2**
Programador responsable: Mario Alexander Meneses Yero	
Descripción: Se realiza el análisis léxico de la rutina IL que se está editando, si existe algún error, será reportado al momento de su detección.	

Tabla 7 Historia de usuario № 3.

Historia de usuario	
Número: 3	**Usuario: Línea de Sistemas Embebidos**
Nombre de historia: Realización del análisis sintáctico a la rutina IL en edición.	
Prioridad del negocio: Alta	**Riesgo en el desarrollo: Alto**
Puntos estimados: 3	**Iteración asignada: 2**
Programador responsable: Mario Alexander Meneses Yero	
Descripción: Se realiza el análisis sintáctico de la rutina IL que se está editando, si existe algún error, será reportado al momento de su detección.	

Tabla 8 Historia de usuario № 4.

Historia de usuario	
Número: 4	**Usuario: Línea de Sistemas Embebidos**
Nombre de historia: Realización del análisis semántico a la rutina IL en edición.	
Prioridad del negocio: Alta	**Riesgo en el desarrollo: Alto**
Puntos estimados: 3	**Iteración asignada: 2**
Programador responsable: Mario Alexander Meneses Yero	
Descripción: Se realiza el análisis semántico de la rutina IL que se está editando, si existe algún error, será reportado al momento de su detección.	

Tabla 9 Historia de usuario № 5.

Historia de usuario	
Número: 5	**Usuario: Línea de Sistemas Embebidos**
Nombre de historia: Detección y reporte de errores presentes en la rutina IL en edición.	

Prioridad del negocio: Alta	**Riesgo en el desarrollo: Medio**
Puntos estimados: 3	**Iteración asignada: 2**
Programador responsable: Mario Alexander Meneses Yero	
Descripción: Debe permitir la detección de errores en la rutina IL que se está editando, tanto el señalamiento en tiempo real, como permitir el uso de una opción llamada “Chequear” o “Construir”, la cual realizará el chequeo correspondiente y devolverá una lista con los errores detectados.	

Tabla 10 Historia de usuario № 6.

Historia de usuario	
Número: 6	**Usuario: Línea de Sistemas Embebidos**
Nombre de historia: Obtener y establecer código editado en el lenguaje IL.	
Prioridad del negocio: Alta	**Riesgo en el desarrollo: Alto**
Puntos estimados: 2	**Iteración asignada: 3**
Programador responsable: Mario Alexander Meneses Yero	
Descripción: El sistema debe de la captura o establecimiento de todo el código IL en edición, para que posteriormente, el mismo sea guardado y cargado en un archivo XML, según el estándar PLCopen.	

Tabla 11 Historia de usuario № 7.

Historia de usuario	
Número: 7	**Usuario: Línea de Sistemas Embebidos**
Nombre de historia: Requisitos de software para la implementación.	
Descripción: Debe permitir su ejecución para todas las plataformas de la familia GNU/Linux y Windows. Instaladas las siguientes bibliotecas: Boost, en su versión 1.46 o superior, para la implementación de la detección de errores.	

Tabla 12 Historia de usuario № 8.

Historia de usuario	
Número: 8	**Usuario: Línea de Sistemas Embebidos**
Nombre de historia: Requisitos de hardware para uso.	
Descripción: Prestaciones como:	

Capacidad de almacenamiento: 25 MB Procesador: 400 MHz Memoria RAM: 128 MB Mínimo, deben de estar presentes para que la aplicación funcione correctamente.

Tabla 13 Historia de usuario № 9.

Historia de usuario	
Número: 9	**Usuario: Línea de Sistemas Embebidos**
Nombre de historia: Delimitantes en el diseño e implementación.	
Descripción: Se definen para el desarrollo de la solución, las siguientes restricciones: Lenguaje de programación: C++ Framework: Qt 5.0.2 Entorno integrado de desarrollo: Qt Creator 3.0.0	

Tabla 14 Historia de usuario № 10.

Historia de usuario	
Número: 10	**Usuario: Línea de Sistemas Embebidos**
Nombre de historia: Características de apariencia o interfaz externa.	
Descripción: La interfaz debe de ser comprensible para su funcionamiento, permitiendo la utilización del sistema sin necesidad de mucho entrenamiento o de adiestramiento en el mismo. Debe poseer uniformidad en las interfaces, así como el diseño y los colores. No deben existir mensajes que posean ambigüedades y todos los colores de la interfaz deben ser claros.	

2.2 Estimación de esfuerzo por historia de usuario.

La tabla que se muestra a continuación, muestra la estimación del esfuerzo por historia (HU) de usuario, según el orden en que aparecen las HU. Esto se hace con la intención de que los programadores obtengan una estimación de dichas historias en cuanto al nivel de detalle, o sea, para fijar el período de tiempo que se puede tardar en la implementación de cada una. Un punto es considerado una semana ideal de trabajo, donde se trabaje todo el tiempo planeado ininterrumpidamente.

Tabla 15 Estimación de esfuerzo por historia de usuario.

Historia de usuario	**Puntos**

	estimados
Edición de la rutina IL.	2
Realización del análisis léxico a la rutina IL en edición.	3
Realización del análisis sintáctico a la rutina IL en edición.	3
Realización del análisis semántico a la rutina IL en edición.	3
Detección y reporte de errores presentes en la rutina IL en edición.	3
Obtener y establecer código editado en el lenguaje IL.	2

2.3 Plan de entregas.

El plan de entregas se elabora una vez se culmina la realización de las HU. Aquí se especifica en que entrega será implementada cada historia de usuario, y se fija una fecha para la culminación de la misma.

Tabla 16 Plan de entregas.

Plan de entregas				
EditorLI				
Fecha de reunión de planificación:		02/12/2014		
Nombre de documentador:		Mario Alexander Meneses Yero		
Entrega №:		3		
Historias de usuario a implementar				
№. HU	**Título**	**Prioridad**	**Fecha en la que se hará entrega**	**Liberación en la que será incluida**
1	Edición de la rutina IL.	Alta	26/03/2014	1
2	Realización del análisis léxico a la rutina IL en edición.	Alta	16/04/2014	2
3	Realización del análisis sintáctico a la rutina IL en edición.	Alta	27/04/2014	2
4	Realización del análisis semántico a la rutina IL en edición.	Alta	06/05/2014	2
5	Detección y reporte de errores presentes en la rutina IL en edición.	Alta	22/05/2014	2
6	Obtener y establecer código editado en	Alta	26/05/2014	3

	el lenguaje IL.			

2.4 Plan de iteraciones.

Iteración 1

Tabla 17 Iteración № 1.

Historias de usuario	Tiempo estimado (semana ideal de trabajo)	Iteración asignada	Tiempo real
Edición de la rutina IL.	2	1	2

Iteración 2

Tabla 18 Iteración № 2.

Historias de usuario	Tiempo estimado (semana ideal de trabajo)	Iteración asignada	Tiempo real
Realización del análisis léxico a la rutina IL en edición.	3	2	3
Realización del análisis sintáctico a la rutina IL en edición.	3	2	3
Realización del análisis semántico a la rutina IL en edición.	3	2	3
Detección de errores presentes en la rutina IL en edición.	3	2	3

Iteración 3

Tabla 19 Iteración № 3.

Historias de usuario	Tiempo estimado (semana ideal de trabajo)	Iteración asignada	Tiempo real
Obtener y establecer código editado en el lenguaje IL.	2	3	2

2.5 Tareas de ingeniería.

Iteración 1

Tabla 20 Planificación para la primera iteración.

Historia de usuario	Tareas de ingeniería (№)	Puntos estimados
Edición de la rutina IL.	1. Diseño e implementación del prototipo interfaz EditorLI.(0.5) 2. Implementación de la clase codeeditor.(1) 3. Implementación de la clase highlighter.(1) 4. Implementación de la clase editorli.(0.5)	2

Las tareas de ingeniería para esta iteración son:

Tabla 21 Tarea de ingeniería № 1.

Tarea	
Número tarea: 1	**Número HU: 1**
Nombre tarea: Diseño e implementación del prototipo interfaz EditorLI.	
Tipo de tarea: Desarrollo	**Puntos estimados: 0.5**
Fecha inicio:10/03/2014	**Fecha fin:13/03/2014**
Programador responsable: Mario Alexander Meneses Yero	
Descripción: Se realiza el diseño del prototipo interfaz para la edición del archivo IL en edición.	

Tabla 22 Tarea de ingeniería № 2.

Tarea	
Número tarea: 2	**Número HU: 1**
Nombre tarea: Implementación de la clase codeeditor.	
Tipo de tarea: Desarrollo	**Puntos estimados:1**
Fecha inicio:13/03/2014	**Fecha fin:17/03/2014**
Programador responsable: Mario Alexander Meneses Yero	
Descripción: Se encarga de la de la estructura así como el completamiento automatizado o con la opción “CTRL+SPACE”.	

Tabla 23 Tarea de ingeniería № 3.

Tarea	
Número tarea: 3	**Número HU:1**
Nombre tarea: Implementación de la clase highlighter.	
Tipo de tarea: Desarrollo	**Puntos estimados:1**
Fecha inicio: 17/03/2014	**Fecha fin: 22/03/2014**
Programador responsable: Mario Alexander Meneses Yero	
Descripción: Es la encargada del resaltado de las palabras reservadas del lenguaje, así como los comentarios.	

Tabla 24 Tarea de ingeniería № 4.

Tarea	
Número tarea: 4	**Número HU: 1**
Nombre tarea: Implementación de la clase editorli.	
Tipo de tarea: Desarrollo	**Puntos estimados: 0.5**
Fecha inicio: 22/03/2014	**Fecha fin: 26/03/2014**
Programador responsable: Mario Alexander Meneses Yero	
Descripción: La clase se encarga de orientar las acciones que el operador va realizando en el prototipo interfaz de edición de la rutina IL.	

Iteración 2

Tabla 25 Planificación para la segunda iteración.

Historia de usuario	Tareas de ingeniería (№)	Puntos estimados
Realización del análisis léxico a la rutina IL en edición.	5. Definición de los elementos fundamentales del lenguaje Lista de Instrucciones.(0.5) 6. Implementación del análisis léxico.(2) 7. Enlace del análisis léxico con la clase controladora.(0.5)	3
Realización del análisis sintáctico a la rutina IL en edición.	8. Definición de la gramática a utilizar según el lenguaje Lista de Instrucciones.(0.2) 9. Conversión de la gramática a PEG	3

	(*Parsing Expression Grammar*).(0.3) 10. Implementación del análisis sintáctico.(2) 11. Enlace del análisis sintáctico con la clase controladora.(0.5)	
Realización del análisis semántico a la rutina IL en edición.	12. Implementación del análisis semántico.(2.5) 13. Enlace del análisis semántico con la clase controladora.(0.5)	3
Detección y reporte de errores presentes en la rutina IL en edición.	14. Implementación de la clase controladora, permitiendo la respuesta de encontrarse algún error.(2.5) 15. Enlace de la clase controladora con el prototipo interfaz del sistema.(0.5)	3

Las tareas de ingeniería para esta iteración son:

Tabla 26 Tarea de ingeniería № 5.

Tarea	
Número tarea: 5	**Número HU: 2**
Nombre tarea: Definición de los elementos fundamentales del lenguaje Lista de Instrucciones.	
Tipo de tarea: Desarrollo	**Puntos estimados: 0.5**
Fecha inicio:26/03/2014	**Fecha fin: 29/03/2014**
Programador responsable: Mario Alexander Meneses Yero	
Descripción: Se seleccionan los elementos del lenguaje Lista de Instrucciones que formarán parte del proceso de traducción. Se especifican las expresiones regulares que indican el conjunto de posibles secuencias de caracteres definidas.	

Tabla 27 Tarea de ingeniería № 6.

Tarea	
Número tarea: 6	**Número HU: 2**
Nombre tarea: Implementación del análisis léxico.	
Tipo de tarea: Desarrollo	**Puntos estimados:2**

Fecha inicio: 29/03/2014	**Fecha fin:11/04/2014**
Programador responsable: Mario Alexander Meneses Yero	
Descripción: Habiendo agrupado los elementos más importantes del lenguaje Lista de Instrucciones, es posible realizar un chequeo en todo el archivo en edición y determinar cuáles caracteres pertenecen a la gramática y cuáles no.	

Tabla 28 Tarea de ingeniería № 7.

Tarea	
Número tarea: 7	**Número HU: 2**
Nombre tarea: Enlace del análisis léxico con la clase Controladora.	
Tipo de tarea: Desarrollo	**Puntos estimados: 0.5**
Fecha inicio: 11/04/2014	**Fecha fin:16/04/2014**
Programador responsable: Mario Alexander Meneses Yero	
Descripción: Al dejar de hacer entradas por un corto lapso de tiempo, se pasara a realizar el análisis léxico a la rutina IL en edición, a través de la instancia de la clase controladora existente en el análisis léxico. Si se detecta algún error, este se muestra inmediatamente en la pestaña “Salida de errores”, ubicada en la parte central inferior de la interfaz, mostrando la línea y se marca la línea donde se encontró el error.	

Tabla 29 Tarea de ingeniería № 8.

Tarea	
Número tarea: 8	**Número HU: 3**
Nombre tarea: Definición de la gramática a utilizar según el lenguaje Lista de Instrucciones.	
Tipo de tarea: Desarrollo	**Puntos estimados: 0.2**
Fecha inicio: 16/04/2014	**Fecha fin: 17/04/2014**
Programador responsable: Mario Alexander Meneses Yero	
Descripción: La gramática define la estructura que tendrá que poseer cada rutina para su programación.	

Tabla 30 Tarea de ingeniería № 9.

Tarea	
Número tarea: 9	**Número HU: 3**

Nombre tarea: Conversión de la gramática a PEG (Parsing Expression Grammar).	
Tipo de tarea: Desarrollo	**Puntos estimados: 0.3**
Fecha inicio: 17/04/2014	**Fecha fin:18/04/2014**
Programador responsable: Mario Alexander Meneses Yero	
Descripción: La gramática utilizada por la biblioteca Boost Spirit Qi (boost::spirit::qi), es un derivado de las gramáticas de libre contexto (EBNF, por sus siglas en inglés); en la cual se especifica cómo será la estructura a seguir por la rutina en edición.	

Tabla 31 Tarea de ingeniería № 10.

Tarea	
Número tarea: 10	**Número HU: 3**
Nombre tarea: Implementación del análisis sintáctico.	
Tipo de tarea: Desarrollo	**Puntos estimados: 2**
Fecha inicio: 18/04/2014	**Fecha fin: 25/04/2014**
Programador responsable: Mario Alexander Meneses Yero	
Descripción: Implementación del análisis sintáctico, encargado del análisis de la gramática, así como las reglas que la componen.	

Tabla 32 Tarea de ingeniería № 11.

Tarea	
Número tarea: 11	**Número HU: 3**
Nombre tarea: Enlace del análisis sintáctico con la clase Controladora.	
Tipo de tarea: Desarrollo	**Puntos estimados: 0.5**
Fecha inicio: 25/04/2014	**Fecha fin: 27/04/2014**
Programador responsable: Mario Alexander Meneses Yero	
Descripción: Al dejar de teclear por un corte espacio de tiempo, luego de que haya concluido el análisis léxico, se realiza el análisis sintáctico. Si se detecta algún error, este se muestra inmediatamente en la pestaña “Salida de errores”, ubicada en la parte central inferior de la interfaz, mostrando la línea y se marca la línea donde se encontró el error.	

Tabla 33 Tarea de ingeniería № 12.

Tarea	
Número tarea: 12	**Número HU: 4**

Nombre tarea: Implementación del análisis semántico.	
Tipo de tarea: Desarrollo	**Puntos estimados: 2.5**
Fecha inicio: 27/04/2014	**Fecha fin: 04/05/2014**
Programador responsable: Mario Alexander Meneses Yero	
Descripción: La realización del análisis semántico comienza partiendo del Árbol de Sintaxis Abstracta (AST), que se genera en la fase de análisis sintáctico.	

Tabla 34 Tarea de ingeniería № 13.

Tarea	
Número tarea: 13	**Número HU: 4**
Nombre tarea: Enlace del análisis semántico con la clase Controladora.	
Tipo de tarea: Desarrollo	**Puntos estimados: 0.5**
Fecha inicio: 04/05/2014	**Fecha fin: 06/05/2014**
Programador responsable: Mario Alexander Meneses Yero	
Descripción: Al dejar de hacer entradas por un corto período de tiempo, y posterior de que se hallan realizado tanto el análisis léxico como el sintáctico, se realiza el análisis semántico, si se detecta algún error, este se muestra inmediatamente en la pestaña “Salida de errores”, ubicada en la parte central inferior de la interfaz, mostrando la línea y se marca la línea donde se encontró el error.	

Tabla 35 Tarea de ingeniería № 14.

Tarea	
Número tarea: 14	**Número HU: 5**
Nombre tarea: Implementación de la clase controladora, permitiendo la respuesta de encontrarse algún error.	
Tipo de tarea: Desarrollo	**Puntos estimados: 1.5**
Fecha inicio: 06/05/2014	**Fecha fin: 20/05/2014**
Programador responsable: Mario Alexander Meneses Yero	
Descripción: Es la encargada de instanciar en todas las fases de análisis, tanto léxico, sintáctico, como semántico, si al terminar de instanciar en cada uno de los anteriores (siguiendo ese orden), no se detecta error en ninguna de las fases anteriores, significa que la rutina IL creada es correcta.	

Tabla 36 Tarea de ingeniería № 15.

Tarea	
Número tarea: 15	**Número HU: 5**
Nombre tarea: Enlace de la clase controladora con el prototipo interfaz del sistema.	
Tipo de tarea: Desarrollo	**Puntos estimados: 0.5**
Fecha inicio: 20/05/2014	**Fecha fin:22/05/2014**
Programador responsable: Mario Alexander Meneses Yero	
Descripción: Al dejar de teclear por un lapso de tiempo, se procede a realizar el análisis léxico, luego el análisis sintáctico y posterior a este el análisis semántico, si se detecta algún error, este se muestra inmediatamente en la pestaña “Salida de errores”, ubicada en la parte central inferior de la interfaz.	

Iteración 3

Tabla 37 Planificación para la tercera iteración.

Historia de usuario	Tareas de ingeniería (№)	Puntos estimados
Obtener y establecer código editado en el lenguaje IL.	16. Implementación del código para obtener el código editado.(1) 17. Implementación del código para establecer el código editado.(1)	2

Las tareas de ingeniería para esta iteración son:

Tabla 38 Tarea de ingeniería № 16.

Tarea	
Número tarea: 16	**Número HU: 6**
Nombre tarea: Implementación del código para obtener el código editado.	
Tipo de tarea: Desarrollo	**Puntos estimados: 1**
Fecha inicio: 22/05/2014	**Fecha fin: 24/05/2014**
Programador responsable: Mario Alexander Meneses Yero	
Descripción: El sistema contiene una opción, que vincula el texto editado en la rutina IL en ejecución, con el archivo permitiendo donde será guardado, permitiendo su obtención, lo que posibilita posteriormente que la rutina creada sea guardada en un XML según dicta el estándar PLCopen.	

Tabla 39 Tarea de ingeniería № 17.

Tarea	
Número tarea: 17	**Número HU: 6**
Nombre tarea: Implementación del código para establecer el código editado.	
Tipo de tarea: Desarrollo	**Puntos estimados: 1**
Fecha inicio: 24/05/2014	**Fecha fin: 26/05/2014**
Programador responsable: Mario Alexander Meneses Yero	
Descripción: El sistema contiene una opción, que permite establecer un código cargado de otra rutina IL, si el mismo es de tipo IL y fue salvado siguiendo el estándar PLCopen.	

2.6 Propuesta del sistema.

La realización de este trabajo está orientada al desarrollo de una aplicación que permita la programación de rutinas de control al PLC HMI Arex, a través de un prototipo interfaz

creado para la comodidad en el trabajo con las funcionalidades prestadas por la aplicación. El sistema propone la creación de dos módulos, un primer módulo “Interfaz” y un segundo módulo “Analizador”, que en su conjunto conformaran el entorno de desarrollo integrado.

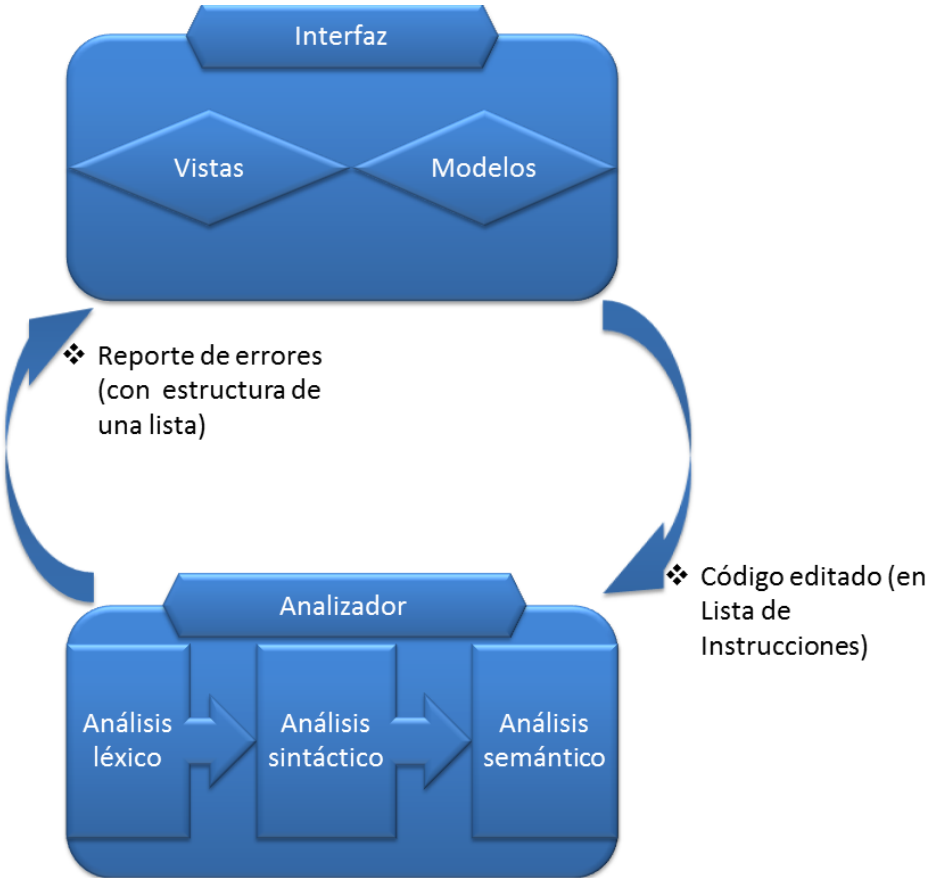


Ilustración 8 Propuesta de solución

2.7 Arquitectura propuesta.

Expresa una organización o esquema estructural fundamental para sistemas software. Proporciona un conjunto de subsistemas predefinidos, especifica sus responsabilidades, e incluye una guía para organizar las relaciones entre ellos.

N capas

La **programación por capas o N capas**, es una arquitectura que, su objetivo primordial, es la separación de la lógica de negocios y la capa de presentación; un ejemplo básico de esto consiste en separar la capa de datos de la capa de presentación al usuario.

La ventaja principal de este estilo es que el desarrollo se puede llevar a cabo en varios niveles y, en caso de que sobrevenga algún cambio, sólo se ataca al nivel requerido sin tener que revisar entre código mezclado.

Además, permite distribuir el trabajo de creación de una aplicación por niveles; de este modo, cada grupo de trabajo está totalmente abstraído del resto de niveles, de forma que basta con conocer la API que existe entre niveles.

En el diseño de sistemas informáticos actual se suelen usar las arquitecturas multinivel o Programación por capas. En dichas arquitecturas a cada nivel se le confía una misión simple, lo que permite el diseño de arquitecturas escalables (que pueden ampliarse con facilidad en caso de que las necesidades aumenten).En la presente solución, se usaron dos capas para su confección, que son:

1. **Capa de presentación:** es la que ve el usuario (también se la denomina "capa de usuario"), presenta el sistema al usuario, le comunica la información y captura la información del usuario en un mínimo de proceso (realiza un filtrado previo para comprobar que no hay errores de formato). También es conocida como interfaz gráfica y debe tener la característica de ser "amigable" (entendible y fácil de usar) para el usuario. Esta capa se comunica únicamente con la capa de negocio.
2. **Capa de análisis:** es donde se lleva a cabo todo el proceso de análisis que tiene lugar en la rutina en edición por el usuario. Esta capa se comunica con la capa de presentación, para recibir las solicitudes y presentar los resultados.

2.8 Patrones de diseño.

La calidad de diseño de la interacción de los objetos y la asignación de responsabilidades presentan gran variación. Las decisiones poco acertadas dan origen a sistemas y componentes frágiles y difíciles de mantener, entender, reutilizar o extender. Una implementación hábil se funda en los principios cardinales que rigen un buen diseño orientado a objetos. En los patrones *GRASP* (*General Responsability Asignment Software Patterns* o Patrones Generales de Software para Asignar Responsabilidades) se codifican algunos de ellos, que se aplican al preparar los diagramas de interacción, cuando se asignan las responsabilidades o durante ambas actividades. En la solución que se presenta a continuación, se tuvo en cuenta la asignación de responsabilidades según los siguientes patrones *GRASP*:

Alta cohesión: El diseño de clases se realizó de forma tal que cada una de ellas tengan responsabilidades moderadas en un área funcional.

Bajo acoplamiento: El acoplamiento es definido por la cantidad de clases relacionadas a una clase, de manera que acoplamiento bajo significa que una clase no depende de muchas clases. Las clases controladoras no dependen de otras clases para realizar sus funcionalidades. Los modelos solo dependen de la clase controladora que lo usa, así se pueden realizar cambios en cada clase de forma independiente.

Creador: Se aplica para la asignación de responsabilidades a las clases relacionadas con la creación de objetos, de forma tal que una instancia de un objeto sólo pueda ser creada por el objeto que contiene la información necesaria para ello. De esta forma una clase controladora puede instanciar objetos, pero es el modelo quien sabe cómo crearlos. En la solución, este patrón se ve representado directamente sobre la clase *codeeditor*.

Controlador: Este patrón sugiere que la lógica de negocios debe estar separada de la capa de presentación, esto para aumentar la reutilización de código y a la vez tener un mayor control. Se ve representado entre la clase *editorli* e *Interpreter* de la solución propuesta.

Experto: Este patrón es el encargado de asignar una responsabilidad al experto en información, o sea, la clase que cuenta con la información necesaria para cumplir la responsabilidad. En la solución, este queda evidenciado en las clases que participan en el proceso de análisis, léxico, sintáctico y semántico.

2.8.1 Patrones GoF (Gang of Four)

Singleton

El patrón de diseño *Singleton* (instancia única) está diseñado para restringir la creación de objetos pertenecientes a una clase o el valor de un tipo a un único objeto. Garantiza que una clase sólo tenga una instancia, y proporciona un punto de acceso global a ella.

El patrón se implementa creando en nuestra clase un método que crea una instancia del objeto sólo si todavía no existe alguna. Para asegurar que la clase no puede ser instanciada nuevamente se regula el alcance del constructor (con atributos como protegido o privado).

En la propuesta de solución, la utilización de este patrón se ve reflejada en la clase *Interpreter*, la cual posee todas las características antes mencionadas.

2.9 Tarjetas Clase-Responsabilidades-Colaboradores (CRC).

Las características más sobresalientes de las tarjetas CRC son su simpleza y ductilidad. Una tarjeta CRC no es más que una ficha de papel o cartón que representa a una entidad del sistema. Estas tarjetas se utilizan para estructurar las clases y a su vez definir las responsabilidades sobre las mismas, así como la simulación de escenarios en el sistema.

Tabla 40 Modelo de tarjeta CRC.

Clase	
Responsabilidades	Colaboradores

A continuación, se definen algunas de las tarjetas CRC del sistema:

Tabla 41 Clase editorli.

editorli	
Se encarga de la construcción de todo el archivo IL.	codeeditor

Tabla 42 Clase codeeditor.

codeeditor	
Crea las condiciones para la edición de la rutina IL.	QComplete Highlighter

Tabla 43 Clase Interpreter.

Interpreter	
Crea el proceso de comunicación entre las clases necesarias en el análisis de la rutina IL en edición. Es la que se define como clase controladora.	ErrorManager InputVar SymbolTable LexerAnalysis SintactycAnalysis SemanticAnalysis

Conclusiones parciales

Se realiza una descripción de las historias de usuario; se cuentan con 10 HU precisadas por el cliente, que serán implementadas en 3 iteraciones. Mediante la descripción de los estándares de codificación y la arquitectura utilizada, se elabora el modelo necesario para llevar a cabo la implementación del sistema. Se ejemplifican algunas de las tarjetas CRC definidas y las tareas de ingeniería, necesarias para establecer el orden de implementación de las HU y del sistema. Se elabora una propuesta de solución, partiendo del flujo actual de eventos.

CAPÍTULO 3: IMPLEMENTACIÓN

Introducción

En el presente capítulo se realiza una descripción de las dos últimas fases de la metodología aplicada. Se describe la selección de las pruebas, así como la realización de las mismas. Por último se describe el proceso de validación que permita comprobar el cumplimiento de los requerimientos del cliente.

3.1 Desarrollo del código implementación

Dentro de la fase de iteraciones se encuentra la implementación o el desarrollo del código el cual se sustenta en buenas prácticas planteadas por el ciclo de vida de XP para esta etapa entre ella se encuentran: disponibilidad del cliente, uso de estándares, programación dirigida por las pruebas (Test-driven programming), programación en pares, integraciones permanentes y ritmo sostenido.

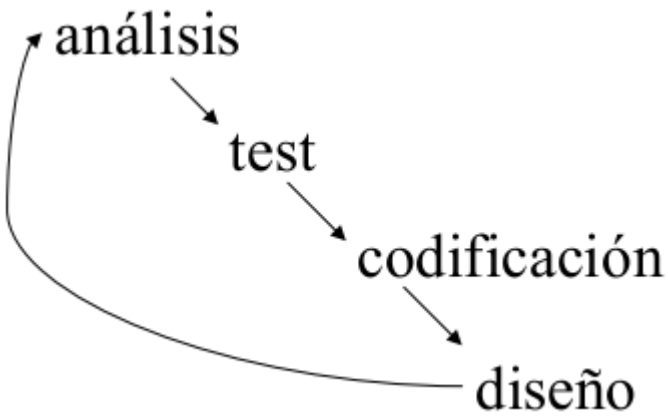


Ilustración 9 Ciclo de implementación de la metodología XP.

En la figura anterior, se muestra el ciclo de implementación dentro de la fase de iteraciones de la metodología XP, la codificación de cada tarea de programación debe de estar guiada por el flujo mostrado. Lo primero a realizar es el análisis de la tarea a realizar, luego, la construcción de la prueba que satisfaga las exigencias de la tarea, seguidamente se codifica para satisfacer la prueba, y a partir de esto se revisa el diseño y se realiza un análisis del mismo.

3.2 Pruebas

La metodología XP propone la aplicación de un tipo de prueba: las de aceptación. Las pruebas de aceptación son utilizadas para evaluar si al final de la iteración se obtuvo la funcionalidad deseada por parte del cliente. El desarrollo dirigido por pruebas (*TDD*, pos sus siglas en inglés), es una práctica de programación que involucra otras dos prácticas: escribir las pruebas primero y refactorización (*Refactoring*).

3.2.1 Pruebas de aceptación

Las pruebas de aceptación son creadas en base a las HU, en cada ciclo de la iteración del desarrollo. El cliente debe especificar uno o diversos escenarios para comprobar que una HU ha sido correctamente implementada. Las pruebas de aceptación son consideradas como “pruebas de caja negra”, los clientes son responsables de verificar que los resultados de estas pruebas sean correctos.

Las pruebas de aceptación son más importantes que las pruebas unitarias dado que significan la satisfacción del cliente con el producto desarrollado y el final de una iteración y el comienzo de la siguiente, por esto el cliente es la persona adecuada para diseñar las pruebas de aceptación. Las pruebas de aceptación son consideradas como “pruebas de caja negra”.

Tabla 44 Caso de prueba de aceptación № 1.

Caso de prueba de aceptación	
Número: 1	Historia de usuario: 1
Nombre: Edición de la rutina IL.	
Descripción: Esta prueba se realiza con el objetivo de comprobar que se permita editar una rutina en el lenguaje Lista de Instrucciones (IL).	
Condiciones de ejecución: ✓ Debe haberse creado un archivo IL.	
Entradas/Pasos de ejecución: ✓ Luego de haber creado el archivo IL, hacer doble clic sobre el mismo para permitir su edición.	
Resultado esperado: Al cargar el editor para el leguaje IL, este debe permitir resaltado de las palabras reservadas del lenguaje, numeración de la línea, completamiento de código, indicador de errores en tiempo real, etc.	

Evaluación de la prueba: No se detectaron no conformidades.

Tabla 45 Caso de prueba de aceptación № 2.

Caso de prueba de aceptación	
Número: 2	Historia de usuario: 2
Nombre: Edición de una rutina IL que contiene errores léxicos.	
Descripción: La siguiente prueba se realiza a fin de conocer si el editor es capaz de gestionar los errores léxicos exitosamente.	
Condiciones de ejecución: ✓ Debe de estar en ejecución el editor IL.	
Entradas/Pasos de ejecución: ✓ Puede no hacer entradas o “teclear” en un lapso entre 1 o 2 segundos.	
Resultado esperado: Al dejar de teclear por al menos 2 segundos, se debe de subrayar los caracteres inválidos, así como mostrar el tipo de error y la línea a la que corresponde en la ventada de salida de errores.	
Evaluación de la prueba: No se detectaron no conformidades.	

Tabla 46 Caso de prueba de aceptación № 3.

Caso de prueba de aceptación	
Número: 3	Historia de usuario:2
Nombre: Edición de una rutina IL que no contiene errores léxicos.	
Descripción: La siguiente prueba se realiza a fin de conocer si el editor es capaz de gestionar los errores léxicos exitosamente.	
Condiciones de ejecución: ✓ Debe de estar en ejecución el editor IL.	
Entradas/Pasos de ejecución: ✓ Al dejar de teclear por al menos 2 segundos.	
Resultado esperado: No se debe de mostrar ningún error.	
Evaluación de la prueba: No se detectaron no conformidades.	

Tabla 47 Caso de prueba de aceptación № 4.

Caso de prueba de aceptación	
Número: 4	Historia de usuario: 3

Nombre: Edición de una rutina IL que contiene errores sintácticos.
Descripción: La siguiente prueba se realiza a fin de conocer si el editor es capaz de gestionar los errores sintácticos exitosamente.
Condiciones de ejecución: ✓ Debe de estar en ejecución el editor IL.
Entradas/Pasos de ejecución: ✓ Dejar de teclear por un espacio de tiempo de al menos 2 segundos.
Resultado esperado: Al dejar de teclear por al menos 2 segundos, se debe de subrayar la sentencia incorrecta, así como mostrar el tipo de error y la línea a la que corresponde en la ventada de salida de errores.
Evaluación de la prueba: No se detectaron no conformidades.

Tabla 48 Caso de prueba de aceptación № 5.

Caso de prueba de aceptación	
Número: 5	Historia de usuario: 3
Nombre: Edición de una rutina IL que no contiene errores sintácticos.	
Descripción: La siguiente prueba se realiza a fin de conocer si el editor es capaz de gestionar los errores sintácticos exitosamente.	
Condiciones de ejecución: ✓ Debe de estar en ejecución el editor IL.	
Entradas/Pasos de ejecución: ✓ Dejar de teclear por un lapso de tiempo de al menos 2 segundos.	
Resultado esperado: No se debe de mostrar ningún error.	
Evaluación de la prueba: No se detectaron no conformidades.	

Tabla 49 Caso de prueba de aceptación № 6.

Caso de prueba de aceptación	
Número: 6	Historia de usuario: 4
Nombre: Edición de una rutina IL que contiene errores semánticos.	
Descripción: La siguiente prueba se realiza a fin de conocer si el editor es capaz de gestionar los errores semánticos exitosamente.	
Condiciones de ejecución: ✓ Debe de estar en ejecución el editor IL.	

Entradas/Pasos de ejecución: ✓ Dejar de teclear por al menos 2 segundos.
Resultado esperado: Al dejar de teclear por al menos 2 segundos, se debe de subrayar el error, así como mostrar el tipo de error y la línea a la que corresponde en la ventada de salida de errores.
Evaluación de la prueba: No se detectaron no conformidades.

Tabla 50 Caso de prueba de aceptación № 7.

Caso de prueba de aceptación	
Número: 7	Historia de usuario: 4
Nombre: Edición de una rutina IL que no contiene errores semánticos.	
Descripción: La siguiente prueba se realiza a fin de conocer si el editor es capaz de gestionar los errores semánticos exitosamente.	
Condiciones de ejecución: ✓ Debe de estar en ejecución el editor IL.	
Entradas/Pasos de ejecución: ✓ Dejar de teclear por un corto espacio de tiempo.	
Resultado esperado: No se debe de mostrar ningún error.	
Evaluación de la prueba: No se detectaron no conformidades.	

Tabla 51 Caso de prueba de aceptación № 8.

Caso de prueba de aceptación	
Número: 8	Historia de usuario: 5
Nombre: Detección y reporte de errores para una rutina IL que contiene errores.	
Descripción: La siguiente prueba se realiza con el objetivo de conocer si el editor es capaz de gestionar los errores existentes.	
Condiciones de ejecución: ✓ Debe de estar en ejecución el editor IL.	
Entradas/Pasos de ejecución: ✓ Dejar de teclear por un corto lapso de tiempo.	
Resultado esperado: Al dejar de teclear, deben de ser subrayados todos los errores presentes en la rutina IL, así como reportar el tipo de cada uno y la línea en la que se encuentra la salida de errores.	

Evaluación de la prueba: No se detectaron no conformidades.

Tabla 52 Caso de prueba de aceptación № 9.

Caso de prueba de aceptación	
Número: 9	Historia de usuario: 5
Nombre: Detección y reporte de errores para una rutina IL que no contiene errores.	
Descripción: La siguiente prueba se realiza con el objetivo de conocer si el editor es capaz de gestionar los errores existentes.	
Condiciones de ejecución: ✓ Debe de estar en ejecución el editor IL.	
Entradas/Pasos de ejecución: ✓ Dejar de teclear por un corto período de tiempo.	
Resultado esperado: No se debe de mostrar ningún error.	
Evaluación de la prueba: No se detectaron no conformidades.	

Tabla 53 Caso de prueba de aceptación № 10.

Caso de prueba de aceptación	
Número: 10	Historia de usuario: 6
Nombre: Obtener el código editado en lenguaje IL.	
Descripción: La siguiente prueba se realiza con el objetivo de conocer si el proceso de obtener el código editado en el lenguaje IL funciona correctamente.	
Condiciones de ejecución: ✓ Debe de existir un proyecto creado. ✓ Debe de existir al menos una rutina IL creada.	
Entradas/Pasos de ejecución: ✓ Hacer “clic” en el botón “Obtener”.	
Resultado esperado: El programa obtiene el código editado de la rutina IL en ejecución.	
Evaluación de la prueba: No se detectaron no conformidades.	

Tabla 54 Caso de prueba de aceptación № 11.

Caso de prueba de aceptación	
Número: 11	Historia de usuario: 6
Nombre: Establecer el código obtenido en una rutina IL.	

Descripción: La siguiente prueba se realiza con el objetivo de conocer si el proceso de establecer el código editado en el lenguaje IL, obtenido de archivo, se inserta correctamente en una rutina IL.
Condiciones de ejecución: ✓ Debe de existir un proyecto creado. ✓ Debe de existir al menos una rutina IL creada.
Entradas/Pasos de ejecución: ✓ Hacer “clic” en el botón “Establecer código”.
Resultado esperado: El programa establece el código editado de archivo, en la rutina IL en ejecución.
Evaluación de la prueba: No se detectaron no conformidades.

Resumen de las pruebas

En la siguiente gráfica, se muestran los resultados obtenidos luego de haber ejecutado las pruebas de aceptación en la iteración correspondiente a cada una. Las no conformidades detectadas en cada iteración, fueron solucionadas antes de dar paso a la siguiente iteración.

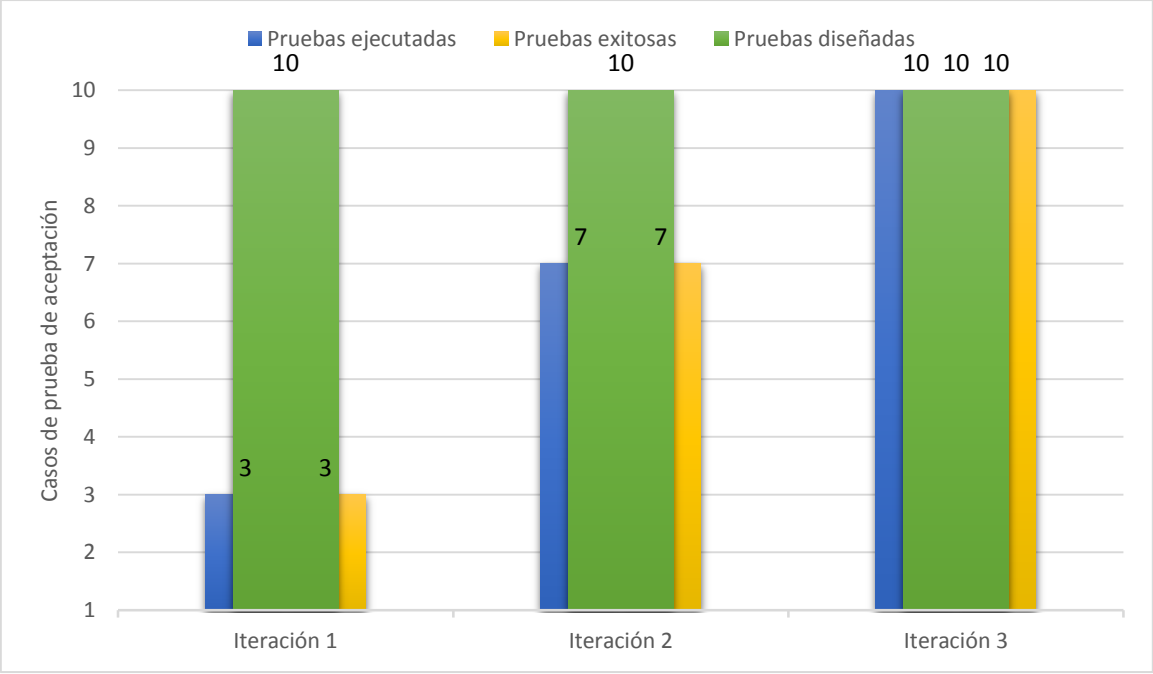


Ilustración 10 Resumen de las pruebas realizadas.

Conclusiones parciales

Se ejecutaron las pruebas de aceptación, mostrando los resultados en una gráfica en la cual se ve el nivel de aceptación en los resultados de las mismas. Al finalizar las pruebas de aceptación, se pudo determinar el grado de satisfacción del cliente al comprobar que la aplicación creada cumple con todos los requisitos especificados.

CONCLUSIONES

Al finalizar el presente trabajo se concluyó que:

- El diseño y desarrollo de una aplicación capaz de realizar la edición y análisis de funciones implementadas en el lenguaje Lista de Instrucciones, respetando las definiciones establecidas en el estándar internacional IEC 61131-3. La aplicación “EditorLI” facilita la programación de subrutinas de control en el lenguaje Lista de Instrucciones.
- Mediante el estudio de un conjunto de herramientas, de características similares, fue posible la definición de los elementos y especificaciones de las instrucciones a implementar para el sistema.
- Se utilizó la metodología XP como guía en el proceso de desarrollo del producto. Las herramientas y tecnologías seleccionadas fueron: *UML* como lenguaje de modelado, el *Visual Paradigm for UML* como herramienta *CASE*, como lenguaje de programación C++, como marco de trabajo *Qt* e IDE de desarrollo *Qt Creator*.
- Las pruebas de aceptación realizadas a la aplicación desarrollada, contribuyeron a determinar que la aplicación “EditorLI” cumple satisfactoriamente con los requerimientos previstos por el cliente.

RECOMENDACIONES

- Integrar la solución desarrollada a la suite Arexsoft.
- Adicionar las funcionalidades restantes del estándar internacional IEC 61131-3, como son, los tipos de datos STRING, DATE, TIME, que no fueron implementados para esta versión de la solución.

BIBLIOGRAFÍA

1. **Amaro Calderón, Sarah Dámaris y Valverde Rebaza, Jorge Carlos. 2010.** *Metodologías Ágiles.* Trujillo : s.n., 2010.
2. **Blanchard, David. 2012.** DBlanchard space. [En línea] 2012. [Citado el: 4 de diciembre de 2013.] http://blanchardspace.wordpress.com/2013/05/06/introduccion-a-c-que-es/.
3. **BOOST. 2012.** Spirit. [En línea] 2012. [Citado el: 5 de dicembre de 2013.] http://boost-spirit.com/home/.
4. **Certificación, Asociación Española de Normalización y. 1993.** *Autómatas programables.* C Génova : AENOR, 1993. 3.
5. **Codesys. 2013.** Eaton Automation. [En línea] 06 de 2013. [Citado el: 09 de 12 de 2013.] http://www.microinnovation.com/en/desktopdefault.aspx/tabid-4/5_view-80/.
6. **Contact, Phenix. 2009.** Moeller. [En línea] 2009. [Citado el: 5 de diciembre de 2013.] http://www.moeller.com.au/easySoft_CoDeSys.asp.
7. **Daniel Vargas Vela, Fernando Martinez Santa.** Diseño e implentación del compilador STL para el PLC-UD. *Diseño e implentación del compilador STL para el PLC-UD.*
8. **2011.** e4800. [En línea] 2011. [Citado el: 9 de 12 de 2013.] http://e4800-plc.blogspot.com/2011/06/plc-defination-by-nema.html.
9. **Eelco van der Wal, Hans Peter Otto. 2009.** *Formato XML para IEC 61131-3.* 2009. 2.01.
10. **Electronics, Schneider. 2012.** Scribd. [En línea] 2012. [Citado el: 7 de 12 de 2013.] http://es.scribd.com/doc/175910213/Indralogic-1-Operating-and-Programming.
11. **Fernández Escribano, Gerardo. 2002.** *Introducción a Extreme Programing.* 2002.
12. **Hernández, Domingo. 2012.** *Introducción al UML.* 2012.
13. **Inc, Beremiz. 2013.** Beremiz. [En línea] 2013. [Citado el: 10 de diciembre de 2013.] http://www.beremiz.org/.

14. **Inc, Twido. 2012.** software.informer. [En línea] 2012. [Citado el: 5 de 12 de 2013.] http://twidosoft.software.informer.com/.

15. *Introducción al Qt y al Qt Creator.* **Vivas Albán, Oscar Andrés. 2010.** Cauca : s.n., 2010.

16. **Programmers, Free. 2010.** Code Project. [En línea] 2010. [Citado el: 10 de diciembre de 2013.] http://www.codeproject.com/Articles/8516/An-Introduction-to-the-Boost-Spirit-Parser-framewo.

17. **Rocha, Daniel. 2012.** [En línea] 2012. [Citado el: 7 de diciembre de 2013.] http://www.slideshare.net/dcrocha/qt-technical-presentation.

18. **Ruiz Catalán, Jacinto. 2010.** *Compiladores: teoría e implementación.* 2010.

19. **Schmuller, Joseph. 2011.** *Aprendiendo UML.* México : Divisió Computación, 2011.

20. **Sierra, Daniel. 2010.** *¿Qué importancia tienen las Herramientas Case?* 2010.

21. **Stroustrup, Bjarne. 2012.** *The Design and Evolution of C++.* s.l. : Addison-Wesley, 2012. 978-0-201-54330-8.

22. **WAGO. 2013.** NHP Electrical Engineering Products. [En línea] 2013. [Citado el: 3 de 12 de 2013.] http://www.nhp.com.au/Product/Products-and-Services/automation-systems/engineering-visualisation-reporting-software/engineering-software/wago-io-pro-caa759333.

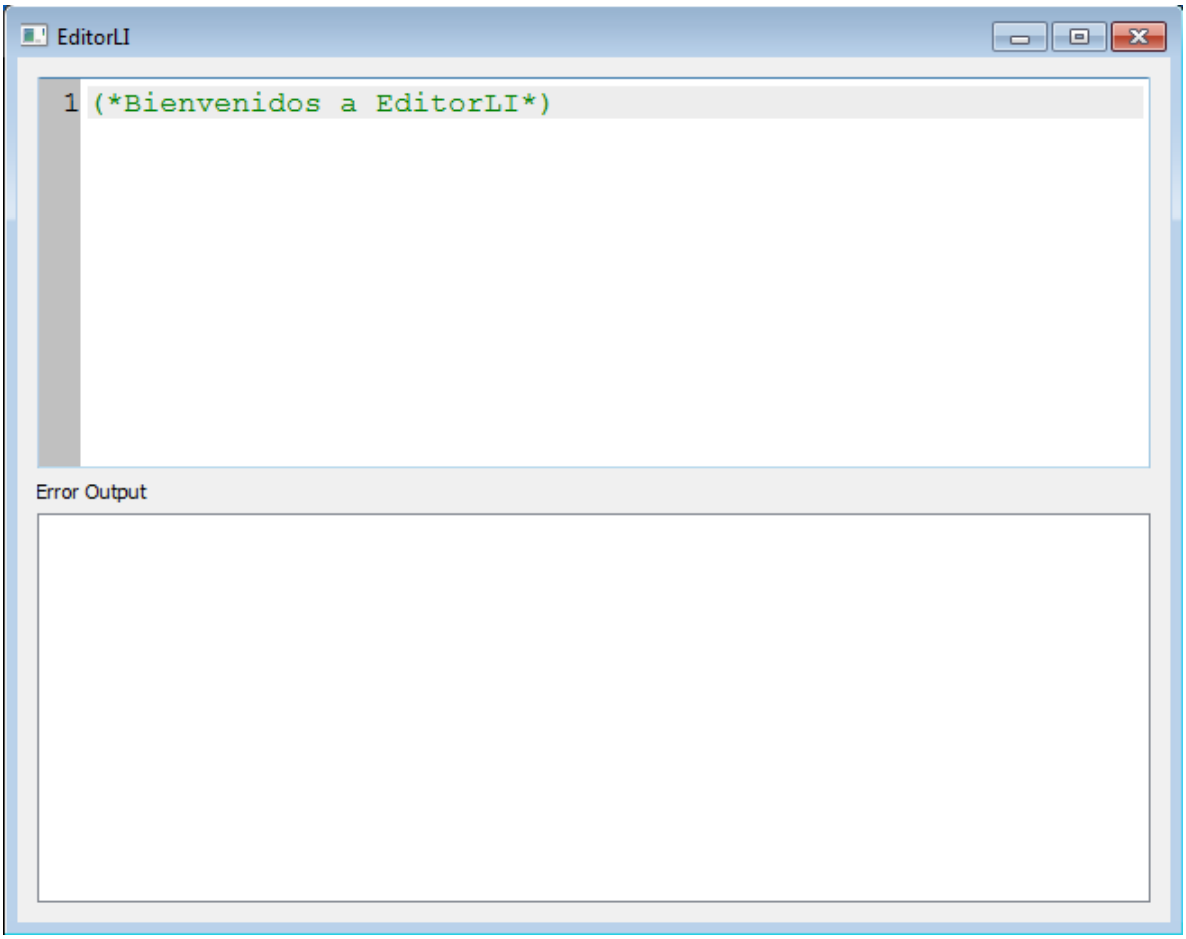
ANEXOS

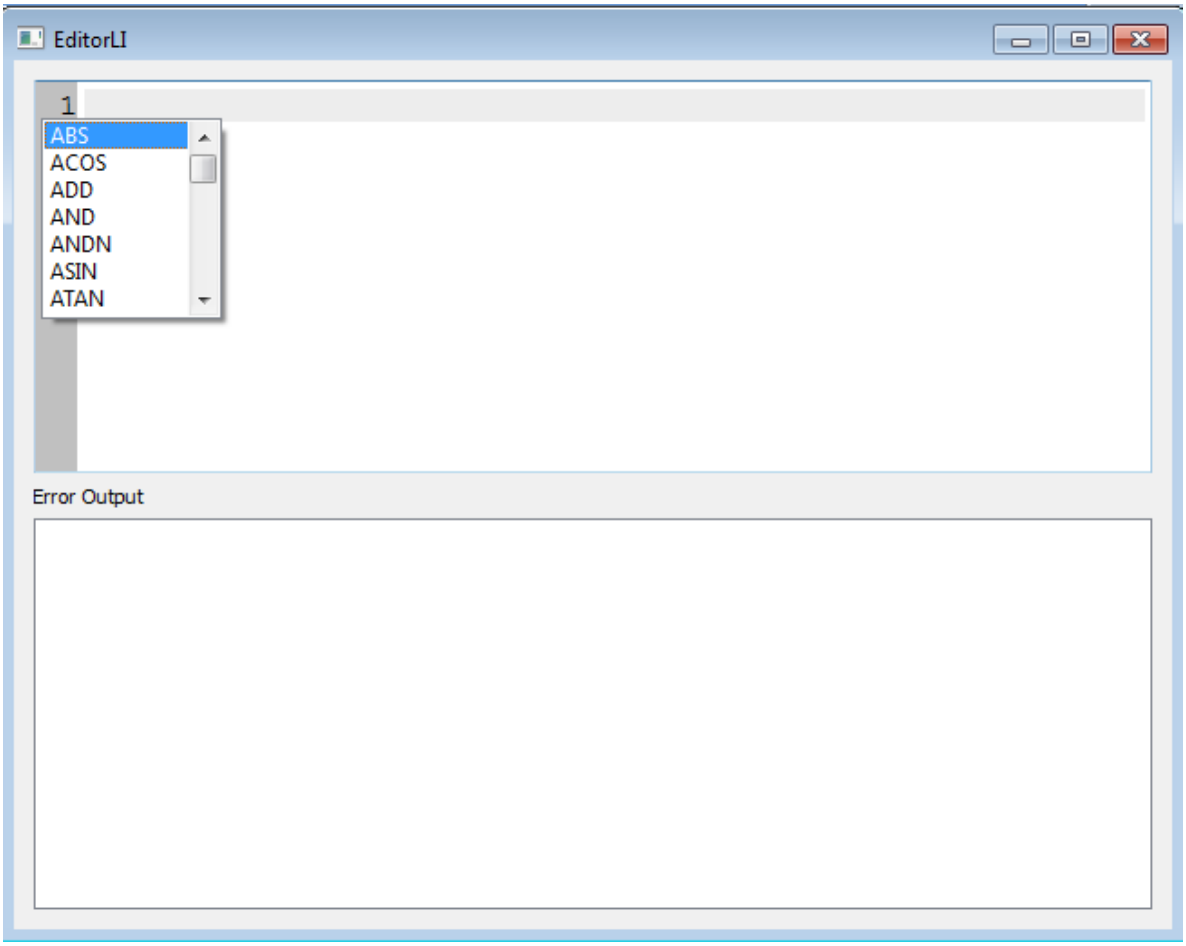
Gramática PEG (Parsing Expression Grammar) definida:

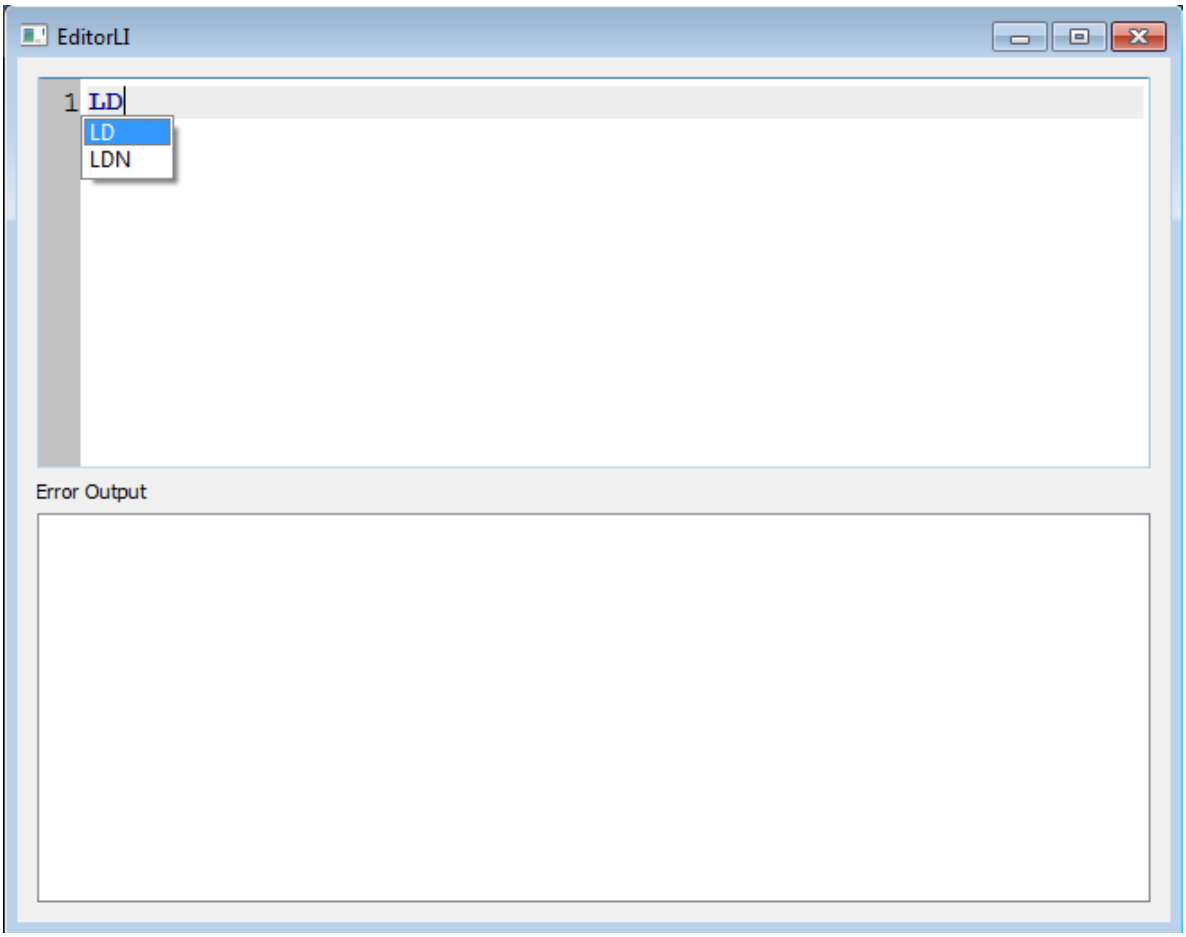
```
program = load >> instruction1;
load = token.n_load [f_oper] >> slot
        | token.load [f_oper] >> slot;
label = - (token.identifier [f_dec_label] >> token.colon);
instructions = label >> instruction;
instruction = bit_logic  >> instruction1
                | mathematics  >> instruction1
                | load  >> instruction1
                | storage  >> instruction1
                | control_logic  >> instruction1
                | contrans >> instruction1
                | logarithmic >> instruction1
                | trigonometric >> instruction1
                | shift >> instruction1
                | comparison  >> instruction1;
instruction1 = - instructions;
bit_logic = token.li_and  [f_oper] >> operand
              | token.n_and [f_oper] >> operand
              | token.li_or [f_oper] >> operand
              | token.n_or  [f_oper] >> operand
              | token.li_xor[f_oper] >> operand
              | token.n_xor [f_oper] >> operand
              | token.reset [f_oper] >> slot
              | token.li_set[f_oper] >> slot;
operand = slot
            | token.par_open
            >> slot
```

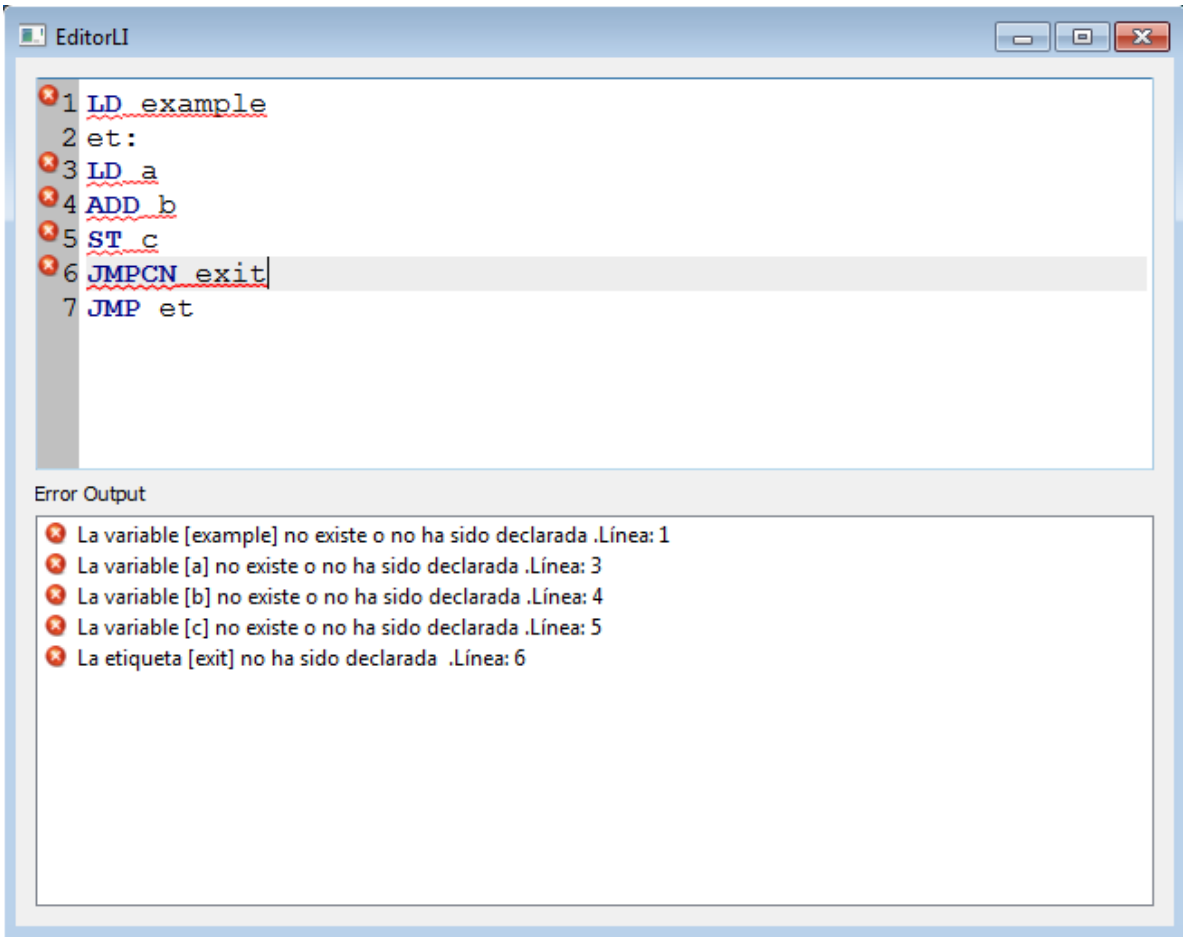
```
                >> nests
                >> token.par_close;
nests = instruction >> nests1;
nests1 = - nests;
slot = token.identifier [f_inst_id];
mathematics = token.add   [f_oper] >> operand
                | token.sub [f_oper] >> operand
                | token.mul [f_oper] >> operand
                | token.div [f_oper] >> operand
                | token.li_mod [f_oper] >> (token.li_float | token.identifier [f_inst_id])
                | token.li_expt [f_oper] >> token.num
                | token.li_move [f_oper] >> token.identifier [f_inst_id];
storage = token.st    [f_oper] >> slot
              | token.n_st [f_oper] >> slot;
control_logic = token.jmp       [f_ctrl_logic_oper]>> token.identifier [f_use_label]
                  | token.cal     [f_call_sub_rut]   >> token.identifier [f_use_label_call]
                  | token.ret
                  | token.jmp_c   [f_ctrl_logic_oper]>> token.identifier [f_use_label]
                  | token.cal_c   [f_call_sub_rut]   >> token.identifier [f_use_label_call]
                  | token.ret_c   [f_ctrl_logic_oper]>> token.identifier [f_use_label]
                  | token.n_jmp_c [f_ctrl_logic_oper]>> token.identifier [f_use_label]
                  | token.n_cal_c [f_call_sub_rut]   >> token.identifier [f_use_label_call]
                  | token.n_ret_c;
contrans = token.trunc [f_oper]
                 | token.real_to_int [f_oper]
                 | token.int_to_real [f_oper];
logarithmic = token.li_abs [f_oper] >> (token.li_float | token.identifier [f_inst_id])
                  | token.li_sqrt [f_oper] >> (token.li_float | token.identifier [f_inst_id])
                  | token.li_ln [f_oper] >> (token.li_float | token.identifier [f_inst_id])
                  | token.li_log [f_oper] >> (token.li_float | token.identifier [f_inst_id])
```

```
                    | token.li_exp [f_oper] >> (token.li_float | token.identifier [f_inst_id]);
trigonometric = token.li_sin [f_oper] >> (token.li_float | token.identifier [f_inst_id])
                        | token.li_cos [f_oper] >> (token.li_float | token.identifier [f_inst_id])
                        | token.li_tan [f_oper] >> (token.li_float | token.identifier [f_inst_id])
                        | token.li_asin [f_oper] >> (token.li_float | token.identifier [f_inst_id])
                        | token.li_acos [f_oper] >> (token.li_float | token.identifier [f_inst_id])
                        | token.li_atan [f_oper] >> (token.li_float | token.identifier [f_inst_id]);
shift = token.li_shl [f_oper] >> token.num
              | token.li_shr [f_oper] >> token.num
              | token.li_ror [f_oper] >> token.num
              | token.li_rol [f_oper] >> token.num;
comparison = token.gt  [f_oper] >> operand
                | token.ge[f_oper] >> operand
                | token.eq[f_oper] >> operand
                | token.ne[f_oper] >> operand
                | token.le[f_oper] >> operand
                | token.lt[f_oper] >> operand;
```


EditorLI
1 (*Bienvenidos a EditorLI*)
Error Output


EditorLI
1
ABS
ACOS
ADD
AND
ANDN
ASIN
ATAN
Error Output


EditorLI
1 LD
LD
LDN
Error Output


EditorLI
1 LD example
2 et:
3 LD a
4 ADD b
5 ST c
6 JMPCN exit
7 JMP et
Error Output
La variable [example] no existe o no ha sido declarada .Línea: 1
La variable [a] no existe o no ha sido declarada .Línea: 3
La variable [b] no existe o no ha sido declarada .Línea: 4
La variable [c] no existe o no ha sido declarada .Línea: 5
La etiqueta [exit] no ha sido declarada .Línea: 6